Mika Alasalmi

Dialogieditori

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

10.5.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Mika Alasalmi Dialogieditori 54 sivua 18.4.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Lehtori Peter Hjort

Insinöörityön tavoitteena oli tehdä pelintekoa helpottava dialogieditori Windows-ympäristössä käytettäväksi. Ohjelman on tarkoitus nopeuttaa dialogidatatiedostojen luontia pelejä varten ja antaa myös koodausta ymmärtämättömille pelintekijöille mahdollisuus luoda näitä tiedostoja.

Ohjelma luotiin C#-ohjelmointikieltä ja .NET-kirjastoa käyttäen Visual Studio -ohjelmankehitysympäristössä. .NET-kirjaston tarjoamista graafisen käyttöliittymän tekemiseen tarkoitetuista työkaluista käytettiin Windows Forms -kirjastoa. Työkaluvalinnat onnistuivat, ja työnteko sujui nopeasti.

Tuotteen vaatimukset ja tuotteen luomien datatiedostojen elementit päätettiin ennen ohjelman tekoa. Valitut elementit ovat nopeakäyttöisiä ja ja auttavat luomaan tiiviitä datatiedostoja, mutta ne myös rajoittavat ohjelman helppokäyttöisyyttä tietyissä rakenteissa, minkä vuoksi tuote ei ole kilpailijoihin verrattuna yhtä monikäyttöinen.

Lopputulos ei täyttänyt kaikkia vaatimuksia, mutta puutteita ei jäänyt paljon, ja kaikki perustavan laatuiset ominaisuudet ovat ohjelmassa. Tulosta voidaan kutsua onnistuneeksi.

Avainsanat	C#, .NET, dialogi, pelinteko

Author Title Number of Pages Date	Mika Alasalmi Dialogue editor 54 pages 18 April 2016
Degree	Bachelor of Engineering
Degree Programme	Information and Communications Technology
Specialisation option	Software Engineering
Instructor	Peter Hjort, Senior Lecturer

The goal of this project was to create a dialogue editor facilitating game development in a Windows environment. The programme is meant to speed up the process of creating dialogue data files for games and to make it possible for people without coding skills to also be able to create such files.

The program was created using the C# programming language and the .NET Framework in a Visual Studio integrated development environment. The Windows Forms library was chosen from the graphical user interface tools provided by the .NET Framework. The chosen tools served the needs of the project well, and the pace of the development was high.

The requirements specification was written and the data elements inside the data files were decided before starting the development phase. The chosen data elements are fast to use and help create compact data files but they are difficult to use when using certain structures which is why some of the competitors' products are more versatile.

The end product did not meet all the requirements but the requirements that were not implemented were few in number and all the fundamental features were implemented. The result can be called a success.

Keywords	C#, .NET, dialogue, game development

Sisällys

1 Johdanto

Insinöörityössä suunnitellaan, toteutetaan ja testataan sovellus, jonka avulla voidaan tuottaa dialogidatatiedostoja pelien käyttöön. Sovelluksen tarkoitus on tarjota Windows-pohjainen graafinen käyttöliittymä dialogidatatiedostojen nopeaan ja tehokkaaseen muodostukseen ja antaa myös koodaustaidottomille kyky valmistaa sellaisia tiedostoja. Tämän lisäksi sovellus on ilmainen, mikä ei ole kilpailevissa tuotteissa yleensä totta.

Sovellus ei ole asiakkaan tilaama, mutta markkinoilla todettiin olevan aukko, johon kuvailun mukainen sovellus mahtuisi, ja siksi se päätettiin toteuttaa.

Raportissa esitellään pelien dialogia, tutkitaan sovelluksen toteutukseen liittyvät valinnat, kuten ohjelmointikielen ja dataformaatin valinta, tarkastellaan, miten toteutus itse asiassa tehtiin ja lopulta testataan lopputulosta muun muassa erillisen testipelin avulla.

2 Pelidialogiteoria

2.1 Dialogipuu ja äärellinen tilakone

Ei-lineaarisista dialogijärjestelmistä niin sanottu dialogipuu on eniten käytetty. Esimerkiksi digitaalisen pelinjakelualusta Steamin kautta 20 eniten myydystä pelistä vuonna 2015 viidessä on ei-lineaarista dialogia, ja jokaisessa näistä viidestä (Witcher 3, Life is Strange, Undertale, Pillars of Eternity ja Fallout 4) on dialogipuu käytössä (1).

Dialogipuu on konsepti, joka on hyvin yleinen eritoten seikkailu- ja roolipeleissä. Dialogipuussa pelaajan annetaan valita, mitä hänen hahmonsa sanoo, ja tämä valinta johtaa uuteen haaraan puussa ja mahdollisesti uusiin valintoihin. Perinteisesti pelaaja tekee valintansa listasta erilaisia vaihtoehtoja, mutta muitakin menetelmiä on ollut käytössä.

Teknisesti dialogipuut eivät välttämättä ole puita ollenkaan. Graafin pitää täyttää tietyt ominaisuudet ollakseen puu, eivätkä ”dialogipuut” aina täytä niitä.

Jos T on graafi ja sillä on n-määrä solmuja, seuraavat lauseet merkitsevät kaikki samaa:

1. T on puu.

2. T:ssä ei ole syklejä, ja sillä on n-1 määrä kaaria.

3. T on yhdistetty, ja sillä on n-1 määrä kaaria.

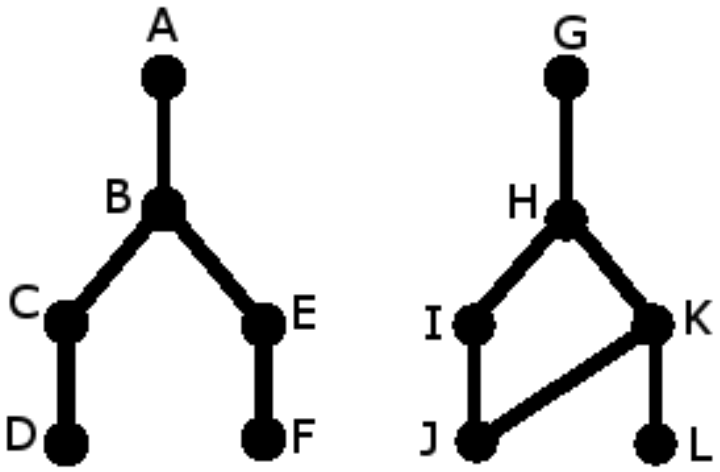
4. T on yhdistetty, ja jokainen kaari on silta.

5. Mitä tahansa kahta solmua yhdistää vain yksi polku. (2, s. 44.)

T:ssä ei ole syklejä, mutta kaaren lisäys mihin tahansa kohtaan graafia synnyttää yhden syklin.

Solmut ovat pisteitä graafeissa, ja kaaret eli viivat taas yhdistävät kahta solmua. Yhdistetyllä tarkoitetaan sitä, että jokaisen kahden solmun välillä on polku. Silta on kaari, jota ilman graafi ei olisi enää yhdistetty.

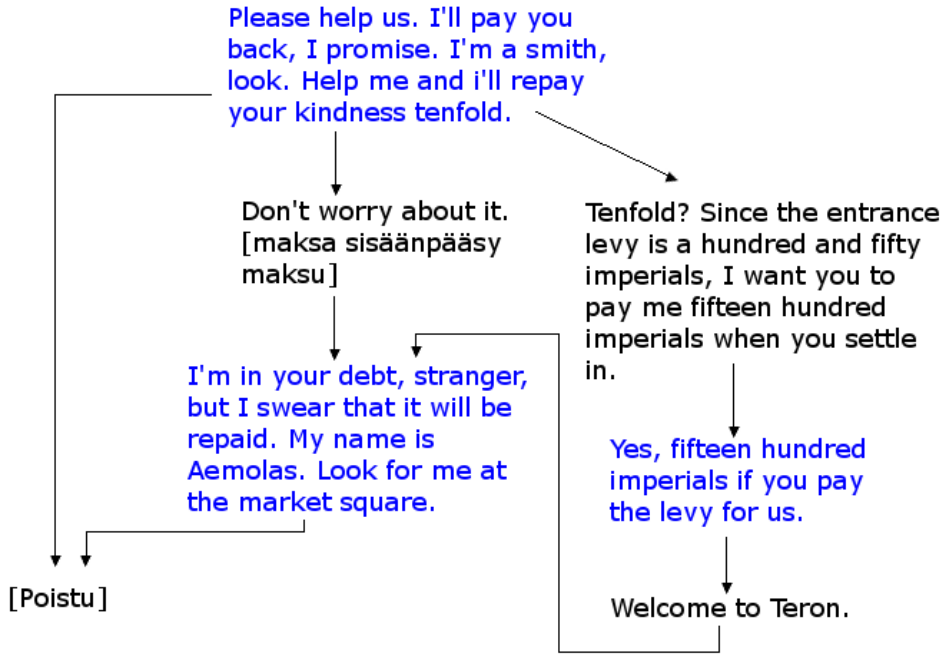
Kuvassa 1 vasemmanpuoleinen graafi on puu, mutta oikeanpuoleinen ei. Vasemmanpuoleisessa kaaria on tasan yksi vähemmän kuin solmuja, ja vastaavasti oikeanpuoleisessa niitä on saman verran. Oikeanpuoleisessa on syntynyt sykli solmuja J:n ja K:n yhdistävän kaaren takia, mutta se on kuitenkin graafi, joka voi vastata pelin ”dialogipuuta” täsmälleen.



Kuva 1. Kaksi graafia, puu ja graafi, joka ei ole puu.

Pelaajan silmissä dialogi voi hyvinkin vastata puumallia, sillä hän ei yhdellä pelikerralla voi mitenkään tietää, että dialogissa on useita reittejä samoihin pisteisiin. Voikin sanoa, että dialogipuu on terminä todellisuutta vastaava ainoastaan tämän illuusion takia.

Kuvan 2 esimerkissä mustat repliikit ovat pelaajan vaihtoehtoja ja siniset NPC:n repliikkejä. NPC on lyhenne englanninkielisestä termistä Non-player character, ja tarkoittaa videopelihahmoa joka ei ole pelaajan hallitsema. Hakasulkeilla merkityt vaihtoehdot tarkoittavat toimintoa, joka tapahtuu dialogin ulkopuolella vaihtoehdon seurauksena. Kuten huomataan, on polkuja, jotka johtavat samoihin repliikkeihin. Pelaajan silmissä puun illuusio säilyy, mutta esiripun takana ei ole puu vaan pikemminkin äärellinen tilakone.



Kuva 2. Eräs ”dialogipuu” pelissä Age of Decadence (3).

Äärellinen tilakone eli äärellinen automaatti on malli, joka määrittää äärellisen joukon tiloja ja sen, kuinka järjestelmä siirtyy tilasta toiseen tiettyjen ehtojen ollessa täyttyessä (4).

Formaalin määrittelyn mukaan äärellinen tilakone on viisikko $(Q, \Sigma, \delta, q_0, F)$, jossa pätee seuraavat lauseet:

1. Q on äärellinen joukko, jota kutsutaan **tiloiksi**.

2. Σ on äärellinen joukko, jota kutsutaan **aakkostoksi**.

3. δ: Q × Σ → Q on **siirtymäfunktio**.

4. $q_0 \in$ Q on **alkutila**.

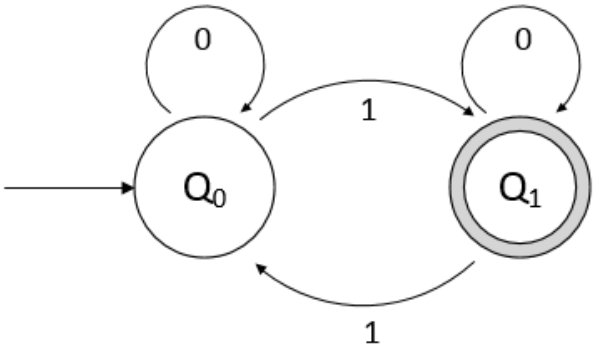
5. F $\subseteq$ Q on hyväksyttyjen tilojen eli **lopputilojen joukko**. (5, s. 35).

Jos A on kaikista koneen M hyväksymistä merkkijonoista koostuva joukko, voidaan sanoa, että tilakoneen M kieli on A, eli L(M) = A. Voidaan myös sanoa, että M tunnistaa A:n.

Tällaista tilakonetta kutsutaan myös deterministiseksi äärelliseksi tilakoneeksi (DFA eli deterministic finite automaton), sillä jokaisessa tilassa on korkeintaan yksi siirtymä kutakin aakkosta kohdin. Toisin sanoen siirtymät ovat yksiselitteisiä eikä vaihtoehtoisia siirtymiä saman aakkosen sisällä ole. Epädeterministiset äärelliset tilakoneet ovat tämän insinöörityön ulkopuolella.

Tilakaavioissa tiloja merkitään ympyröillä, joilla on oma tunniste (kuvassa 3 Q_0 ja Q_1). Kaksoisympyrä tarkoittaa tilaa, joka on lopputila. Nuolet ovat symbolista riippuvia siirtymiä tilasta toiseen, ja tyhjästä tuleva symboliton siirtymä osoittaa alkutilaan.

Kuvan 3 tilakone M hyväksyy vain merkkijonon, joka koostuu ykkösistä ja nollista ja joka sisältää parittoman määrän ykkösiä. Kun tila on Q_0, on ykkösiä parillinen määrä, ja kun tila on Q_1, on ykkösiä pariton määrä. Kun syötemerkkijonon seuraava symboli on 0, pysyy tila samana, ja kun symboli on 1, vaihtaa kone tilaansa (5, s. 41.)



Kuva 3. Yksinkertainen tilakone.

Formaalissa määritelmässä kuvan 3 tilakoneessa M = (Q, Σ, δ, q_0, F) pätevät seuraavat määritykset:

1. Q = { Q_0, Q_1 }
2. Σ = { 0, 1 }
3. δ:ta voidaan kuvailla oheisella siirtymätaulukolla:

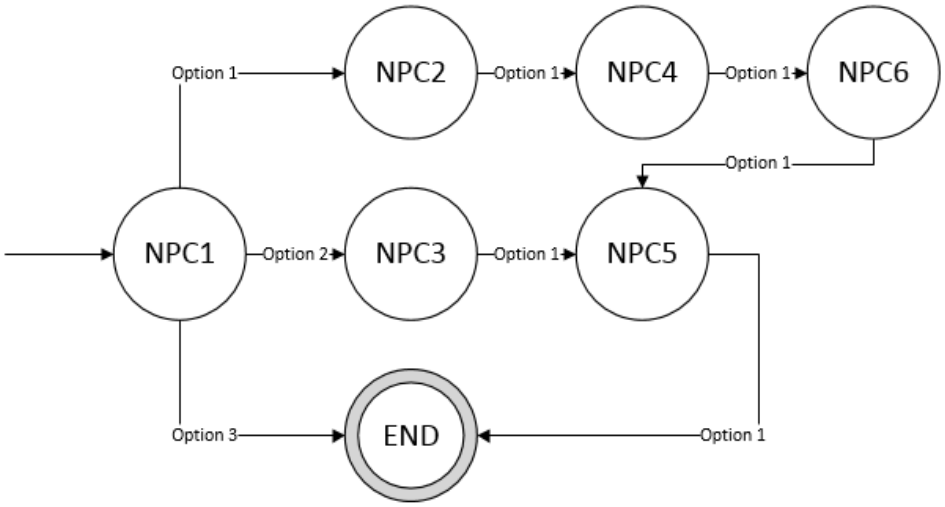
	0	**1**
Q_0	Q_0	Q_1
Q_1	Q_1	Q_0

4. Q_0 on koneen alkutila
5. F = { Q_1 }

A = { w | w koostuu parittomasta määrästä ykkösiä }

L(M) = A

Jos tätä sovelletaan kuvassa 2 esiintyvään dialogipuuhun, voi tuloksena olla kuvan 4 mukainen kaavio.



Kuva 4. Dialogipuusta tilakoneeseen.

Kaaviossa on tilan puutteen vuoksi pelaajan vuorosanat merkitty yksinkertaisesti ”Option”-sanalla ja numerolla, joka ilmaisee vaihtoehdon numeroa tilan sisällä. NPC-vuorosanat ovat tiloja (NPC1, NPC2 jne.). END on lopputila, ja se vastaa dialogipuun [Poistu]-vaihtoehtoa. Kaaviossa ei ole merkitty, mitä tapahtuu, jos NPC2-tilassa annetaan esimerkiksi ”Option 2”-arvo, koska kaavio olisi mennyt kohtuuttoman sotkuiseksi tai kookkaaksi, mutta jos numeroa vastaavaa dialogivaihtoehtoa ei ole, tila pysyy samana ja formaalin määrittelyn siirtymätaulukko sisältää tämän tiedon.

Formaalissa määrittelyssä $M = (Q, \Sigma, \delta, q_0, F)$, jossa pätevät seuraavat määritykset:

1. Q = { NPC1, NPC2, NPC3, NPC4, NPC5, NPC6, END }

2. Σ = { Option 1, Option 2, Option 3 }

3. δ:ta voidaan kuvailla siirtymätaulukolla:

	Option 1	**Option 2**	**Option 3**
NPC1	NPC2	NPC3	END
NPC2	NPC4	NPC2	NPC2
NPC3	NPC5	NPC3	NPC3
NPC4	NPC6	NPC4	NPC4
NPC5	END	NPC5	NPC5
NPC6	NPC5	NPC6	NPC6
END	END	END	END

4. NPC1 on koneen alkutila

5. F = { END }

Dialogipuulla ei tässä dokumentissa tarkoiteta niinkään teknisesti ottaen puuta, vaan dialogitilakonetta, mutta dialogipuu on yleisesti tunnettu termi.

2.2 Pelien dialogi

2.2.1 Varhaishistoria

Dialogin määritys tässä kontekstissa on ”keskustelu kahden tai useamman ihmisen välillä” (6). Tämän insinöörityön yhteydessä ollaan kiinnostuneita ainoastaan ihmisen ja tietokoneen välisestä dialogista peleissä. Yleisesti ottaen tällainen keskustelu tapahtuu pelaajan hahmon ja tietokoneen ohjaaman hahmon eli NPC:n välillä.

Työskennellessään MIT:lle (Massachusetts Institute of Technology) vuonna 1966 Joseph Weizenbaum julkaisi kenties ensimmäisen ohjelman, joka mahdollisti keskustelun ihmisen ja tietokoneen välillä. Tämä ohjelma oli nimeltään ELIZA, ja sen tarkoitus oli tutkia ihmisen ja koneen välistä kommunikaatiota luonnollista kieltä käyttäen. ELIZA oli ja on yksinkertainen avainsanoja ja skriptejä käyttävä ohjelma, mutta se onnistui silti ajoittain luomaan illuusion siitä, että se on ihminen. Tunnetuin ELIZAn yhteydessä käytetty skripti oli luotu mukailemaan psykoterapeutilla käymistä. (7.)

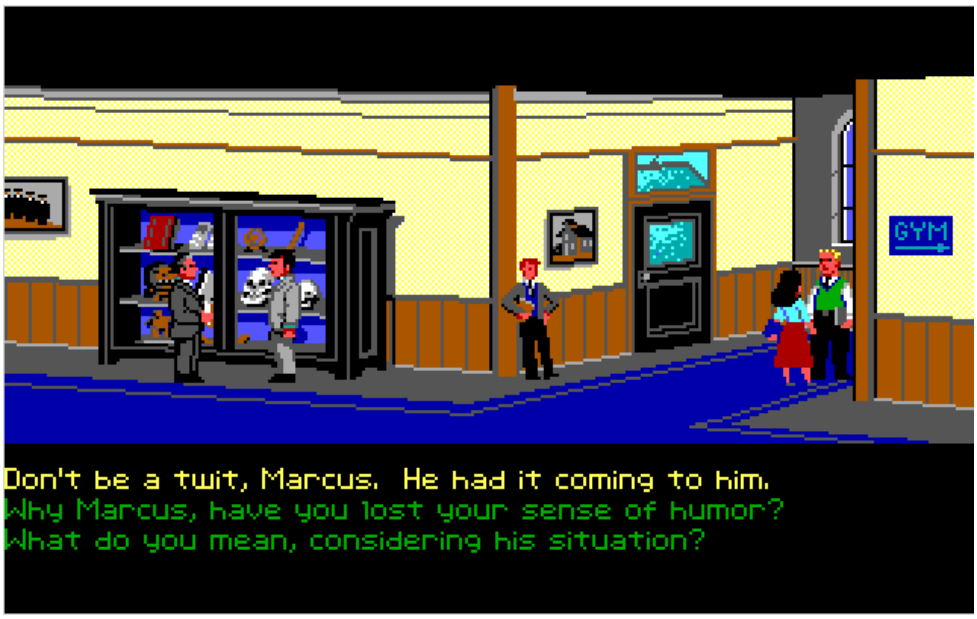
Peleissä dialogijärjestelmät olivat aluksi tekstijäsentimiin ja avainsanoihin pohjautuvia. Tämä oli luonnollinen kehitys siitä, miten tekstiseikkailuissa kirjoitettiin sanoja, jotta hahmo esimerkiksi liikkuisi tai ottaisi esineen maasta. Jo ensimmäisessä tekstiseikkailussa, Colossal Cave Adventuressa vuodelta 1976, oli tällainen järjestelmä (8). Avainsanajärjestelmässä pelaaja voi vapaasti kirjoittaa mitä haluaa, mutta peli tunnistaa vain hyvin harvat sanat. Mahdollisesti kuuluisin esimerkki avainsanadialogista tekstijäsentimen kanssa on Ultima IV: Quest of the Avatar vuodelta 1985 (kuva 5).

Tämäntyyppisissä peleissä dialogi pelaajan osalta käytännössä rajoittui samojen avainsanojen toistamiseen miltei jokaisen vastaantulevan hahmon kohdalla, eikä dialogissa ollut todellista haarautumista, vaan pelaajan sanoo avainsanan, NPC vastaa ja sen jälkeen palataan alkutilanteeseen.

Kuva 5. Dialogijärjestelmä Ultima IV:ssä (9).

Toisto oli keskeinen elementti samalla aikakaudella myös peleissä, jotka käyttivät dialogijärjestelmää, jonka voisi sanoa olevan dialogipuun esi-isä. Esimerkiksi Alice in Wonderland (1985) ja Starflight (1986) (10; 11) olivat pelejä, joissa oli staattiset dialogivaihtoehdot. Käytännössä siis pelaaja pystyi valitsemaan mitä halusi sanoa, mutta nämä vaihtoehdot olivat aina samat riippumatta siitä, kenelle puhuu. Starflightissa tämä oli tehty niin, että ei valittu niinkään, mitä pelaajan hahmo sanoi, vaan pelaajan hahmon asenne tai toiminta. Pelaaja ei nähnyt, mitä pelaaja tarkalleen ottaen sanoi ennen vaihtoehdon valitsemista.

Mahdollisesti ensimmäinen todellinen dialogipuu oli käytössä vuoden 1989 pelissä Indiana Jones and the Last Crusade: The Graphic Adventure (kuva 6). Se oli ensimmäinen SCUMM-pelimoottoria käyttävistä peleistä, jossa käytettiin tällaista dialogijärjestelmää, mutta sen käyttöä jatkettiin myöhemmin muun muassa paljon sitä kuuluisammassa The Secret of Monkey Island -seikkailupelissä (13).



Kuva 6. Dialogipuu Indiana Jones and the Last Crusade -pelissä (12).

2.2.2 Nykypelien dialogimenetelmät

Nykypeleissä dialogi on lähes aina joko lineaarista tai dialogipuuta jossain muodossa käyttävää.

Lineaarisessa dialogissa pelaaja ei voi vaikuttaa dialogiin, vaan dialogi kulkee suoraan käsikirjoituksen mukaan. Pelaaja on siis pelkkä katsoja lineaarisen dialogin aikana, eikä dialogissa ole mitään interaktiivista, joskin joskus pelaaja pystyy esimerkiksi liikkumaan ja tekemään asioita dialogin aikana. Etuna lineaarisessa dialogissa on dialogin luonteva kulku ajoituksen kannalta, sillä taukoja pelaajan miettimisen takia ei synny ollenkaan.

Dialogipuujärjestelmistä moni käyttää kuvan 6 tyyppistä listasta valitsemista, mutta muitakin vaihtoehtoja on. Suosittu menetelmä on niin sanottu dialogipyörä, jonka Bioware jopa patentoi ennen dialogipyörän ensiesiintymistä Mass Effectissä vuonna 2007 (13; 14). Dialogipyörässä dialogivaihtoehdot näytetään nimen mukaisesti eräänlaisen pyörän

ympärillä listan sijasta. Nämä vaihtoehdot ovat lyhennettyjä, eli se, mitä näytöllä näkyy, näyttää vain suuntaa todellisen dialogin sijasta.

Kuvan 7 esimerkissä ”I’ll look into it.” -vaihtoehto on valittuna, mutta tätä vaihtoehtoa klikattaessa hahmo sanookin: ”I’ll see if I can get some answers when I see him.”

Kuva 7. Dialogipyörä Mass Effectissä (14).

Dialogipyörän takana on ajatus siitä, että pitkien dialogivaihtoehtojen lukeminen tuottaa epäluonnollisen temmon dialogiin. Dialogipyörän lyhyiden vaihtoehtojen lukeminen vie paljon vähemmän aikaa, ja dialogin tempo mukailee todellisen elämän keskustelujen tempoa lähemmin.

Joissain peleissä tämä aikatekijä pakotetaan pelaajan vastuulle, sillä pelaajalla on vain rajattu aika valita, mitä hänen hahmonsa sanoo. Yksi esimerkki on kuvan 8 Indigo Prophecy, jossa valitaan dialogipyörän mukaisesti vain lyhyt selitys valinnasta, mutta tämän lisäksi valinnalla on kiire aikarajoituksen vuoksi.

Kuva 8. Dialogivalinnat ja jäljellä oleva aika valinnan tekoon Indigo Prophecyssä (15).

2.3 Kilpailevat tuotteet

Suoraan tämän insinöörityön yhteydessä tehtävän dialogieditorin kanssa kilpailevia tuotteita on markkinoilla hyvin rajallinen määrä. Suurin osa tuotteista, jotka vaikuttavat ensi näkemältä palvelevan samaa tarkoitusta, ei lopulta pysty täyttämään tämän projektin vaatimuksia, kuten esimerkiksi dialogin tallennusta datatiedostoon järkevässä formaatissa. Esimerkiksi Chronicler-työkalu (16) täyttää monet vaatimuksista, mutta tallentaa tiedostoja valitettavasti vain Choice of Gamesin luomaan ChoiceScriptiin.

Unity-pelimoottoria varten on myös luotu monta dialogieditoria, kuten Dialoguer (17), Dialogue System for Unity (18) ja Basic Dialogue Editor (19), mutta tällainen integrointi tiettyyn pelimoottoriin tarkoittaa välttämätöntä käyttökohteiden rajaamista, sillä nämä editorit toimivat vain Unitystä käsin.

Varteenotettavia kilpailijoita löytyi kolme: Articy:draft (20), Chat Mapper (21) ja Talker-MakerDeluxe (22).

2.3.1 Articy:draft – Monimutkainen ohjelma monimutkaisiin tarkoituksiin

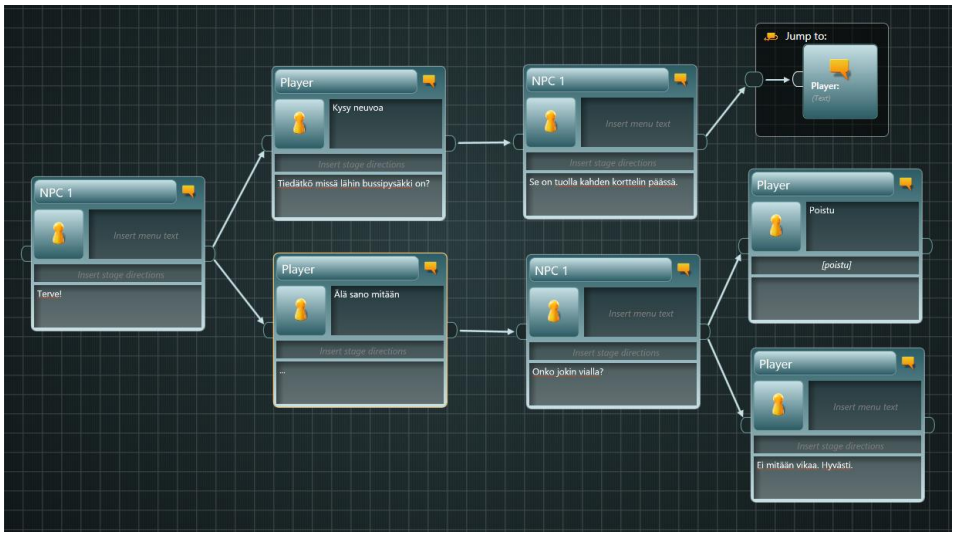
Saksalaisen Nevigon kehittämä Articy:draft on yrityksen kotisivun mukaan vapaasti käännettynä ”yhteistyöympäristö pelisisällön, kuten tehtävien, interaktiivisen dialogin, hahmojen, esineiden ja kenttäasetelmien luomiseen ja organisointiin ensimmäisestä suunnitelmasta suoraan peliin viemiseen asti” (20).

Articy:draft on luultavasti maineikkain kilpailevista tuotteista. Nevigo mainostaa kotisivuillaan asiakkainaan muun muassa isoja pelintekijöitä, kuten Ubisoft, CDProjekt ja Electronic Arts, joskin yritys ei erittele, mitä osia ohjelmasta se käytti. Ohjelma on myös saatavilla Steam-palvelusta.

Articy:draft on kaupallinen tuote, ja lisäksi tuotteen halvemmat versiot eivät salli kaupallisten tuotteiden luomista sen avulla. Halvimmillaan ohjelman uusimmalla versiolla pääsee luomaan kaupallisia tuotteita noin 300 euron hinnalla, mutta tämä on Steam-versio, mikä tarkoittaa, että vain yksi käyttäjä voi käyttää sitä. Usean käyttäjän lisenssi lisää hintaa, useampaa kuin muutamaa käyttäjää tukeva lisenssi on hinnaltaan jo useita tuhansia euroja (20).

Ohjelman sisällä dialogi muodostetaan virtausnäkymässä, johon voidaan sijoittaa dialogielementtejä. Näistä dialogielementeistä tärkeimmät ovat jokseenkin epäintuitiivisia toiminnaltaan: dialogiobjektit eivät ole itsessään repliikkejä, vaikka ne puhekuplilta käyttöliittymässä näyttävätkin, vaan käyttäjän pitää mennä dialogiobjektin sisälle ja lisätä sinne dialogisirpaleita (kuva 9). Dialogisirpaleet sisältävät tiedon muun muassa puhujasta ja

repliikistä. Dialogiobjektit ovat siis oikeastaan keskusteluja. Dialogisirpaleiden välille voidaan muodostaa linkki ja esimerkiksi ehtoja tai toimintoja. Tämä toimii myös dialogiobjektien kohdalla.



Kuva 9. Articy:draftin dialogiobjektin sisällä (20).

Articy:draft on työkalu, joka sisältää paljon ominaisuuksia pitkälti minkä tahansa dialogipuun muodostukseen ja myös monen muunlaisen pelisisällön suunnitteluun ja toteutukseen. Tämän projektin vaatimuksista lähes kaikki täyttyvät ja ovat olemassa muodossa tai toisessa, mutta vakavana puutteena ohjelmasta ei löydy kunnollista tallennusta.

Nevigo on jostain syystä mahdollistanut yksittäisten projektielementtien tallennuksen muissa kuin XML-formaateissa, mutta XML-formaatissa pystyy tallentamaan vain koko projektin. Käytännössä tämä tarkoittaa, että ohjelman luomat XML-tiedostot ovat valtavia ja täynnä tarpeetonta dataa pelkästään dialogipuuta luotaessa. Nevigon Articy:access-kirjastolla voi luoda C++-kielellä oman tallennusmetodin, jossa voi valita tarvittavat osat projektista ja jonka voi tuoda sitten Articy:draftiin valmistuksen jälkeen yhtenä mahdollisena tallennusmahdollisuutena. Articy:access on kuitenkin jälleen muuta kuin ilmainen, ja lisäksi se vaatii monikäyttäjäversion Articy:draftistä.

Kuvassa 10 on esimerkkinä yksi dialogirepliikki ilman Articy:accessia yksinkertaisen testiprojektin XML-tallennuksen tuloksesta. Suurimman osan voisi väittää olevan turhaa tämän projektin tarpeita ajatellen, ja lisäksi mukana on muun muassa HTML-koodia. Todennäköisesti XML-tallennus ei ole tarkoitettu Articy:draftissä suoraan pelin sisälle otettavaksi vaan vain projektitiedoston siirtämiseen koneesta toiseen.

```
<DialogueFragment Id="0x0100000000000015E">
  <DisplayName><![CDATA[NPC 1: "Terve!"]]></DisplayName>
  <Text Count="1" HasMarkup="1">
    <LocalizedString Lang=""><![CDATA[<html><head><style>#s0 {
  </Text>
  <Color>#C8E2E7</Color>
  <TechnicalName>DFr_3A6EE2CC</TechnicalName>
  <ExternalId/>
  <ShortId>0x3A6EE2CC</ShortId>
  <Url>articy://localhost/view/c2b18063-0628-4d76-a207-172a175
  <Features Count="0"/>
  <Pins Count="2">
    <Pin Id="0x0100000000000161" Index="0" Semantic="Input">
      <Expression/>
    </Pin>
    <Pin Id="0x0100000000000162" Index="0" Semantic="Output">
      <Expression/>
    </Pin>
  </Pins>
  <Position X="-14" Y="243"/>
  <Size Width="225" Height="200"/>
  <ZIndex>0</ZIndex>
  <Speaker IdRef="0x010000000000016A"/>
  <StageDirections Count="0"/>
  <MenuText Count="1">
    <LocalizedString Lang=""/>
  </MenuText>
</DialogueFragment>
```

Kuva 10. Yksittäinen repliikki Articy:draftin vakiona tuottamasta XML-tiedostosta (20).

Tallennuksen puutteiden lisäksi ohjelmasta puuttuu tehoa, todennäköisesti juuri ominaisuuksien hajautuneisuuden vuoksi. Monissa kohdin ohjelmaa käyttäjä joutuu luomaan etukäteen asioita tai manuaalisesti tekemään asioita, joitten voisi olettaa luonnollisesti olevan automaattisia. Esimerkiksi dialogirepliikit täytyy aina yhdistää käsin.

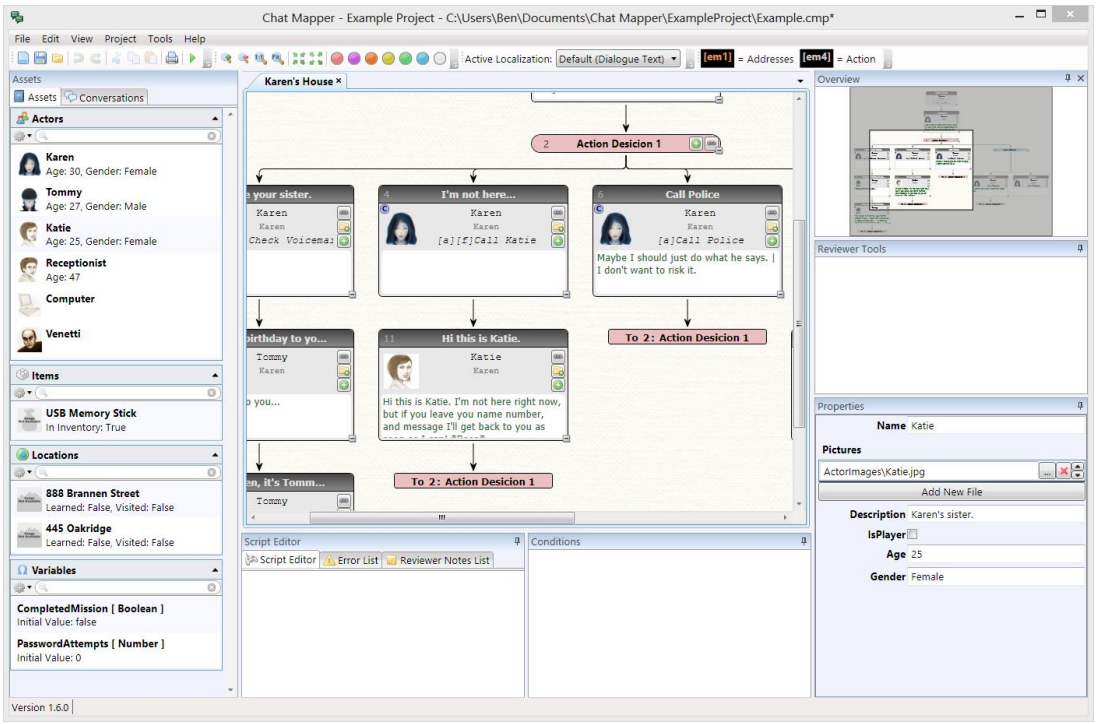
Kuitenkin, jos taloudelliset seikat, tiedoston tallennusmetodin räätälöinnin tarpeellisuus erillisellä ohjelmalla ja käyttöliittymän suhteellinen tehottomuus eivät haittaa, on Articy:draft voimakas ohjelma dialogipuun muodostamiseen.

2.3.2 Chat Mapper – Varteenotettava vaihtoehto

Amerikkalaista Chat Mapperia kuvataan ohjelman kotisivulla yksinkertaisesti seuraavasti: ”epälineaarinen dialogi- ja skeenarioeditori” (21). Chat Mapperistä tarjotaan ilmaista versiota, mutta se on hyvin riisuttu ominaisuuksiltaan (muun muassa tiedoston tallennus XML- tai JSON-formaatissa ei ole mahdollista), eikä sen avulla myöskään ole sallittua luoda kaupallisia sovelluksia. Seuraava taso tästä on editorin Indie-versio, joka sisältää olennaisimmat ominaisuudet ja sallii 100 000 dollarin vuosittaiset tulot ohjelman avustuksella tuotetulla sisällöllä. Indie-versio maksaa 65 dollaria. Rajattomat tulot sallii kaupallinen versio ja julkaisijaversio. Kaupallinen versio sisältää lähes kaikki ominaisuudet ja maksaa 495 dollaria (21).

Chat Mapper on ominaisuuksiltaan ja sitä myötä käyttöliittymältään keskittyneempi dialogiin kuin Articy:draft. Chat Mapperissa kaikki repliikit ovat keskustelujen sisällä, ja keskustelut vastaavat pitkälti dialogiobjekteja Articy:draftissä. Se, että niitä kutsutaan keskusteluiksi dialogiobjektien sijaan, selventää ohjelman käyttöä välittömästi, mutta tämän lisäksi keskustelut ovat selkeästi ylemmän tason olioita ohjelman sisällä.

Ohjelmassa repliikit luodaan myös kätevästi suoraan toisten repliikkien alirepliikeiksi, mikä tarkoittaa, että käyttäjän ei tarvitse manuaalisesti linkittää repliikkejä toisiinsa Articy:draftin tapaan. Chat Mapper poimii myös ensimmäisten valintojen jälkeen kätevään tapaan automaattisesti puhujan ja puhutettavan uusia repliikkejä luodessa, mikä vähentää turhia klikkauksia. Kuvassa 11 yleinen näkymä Chat Mapperin sisällä.



Kuva 11. Chat Mapperin käyttöliittymä.

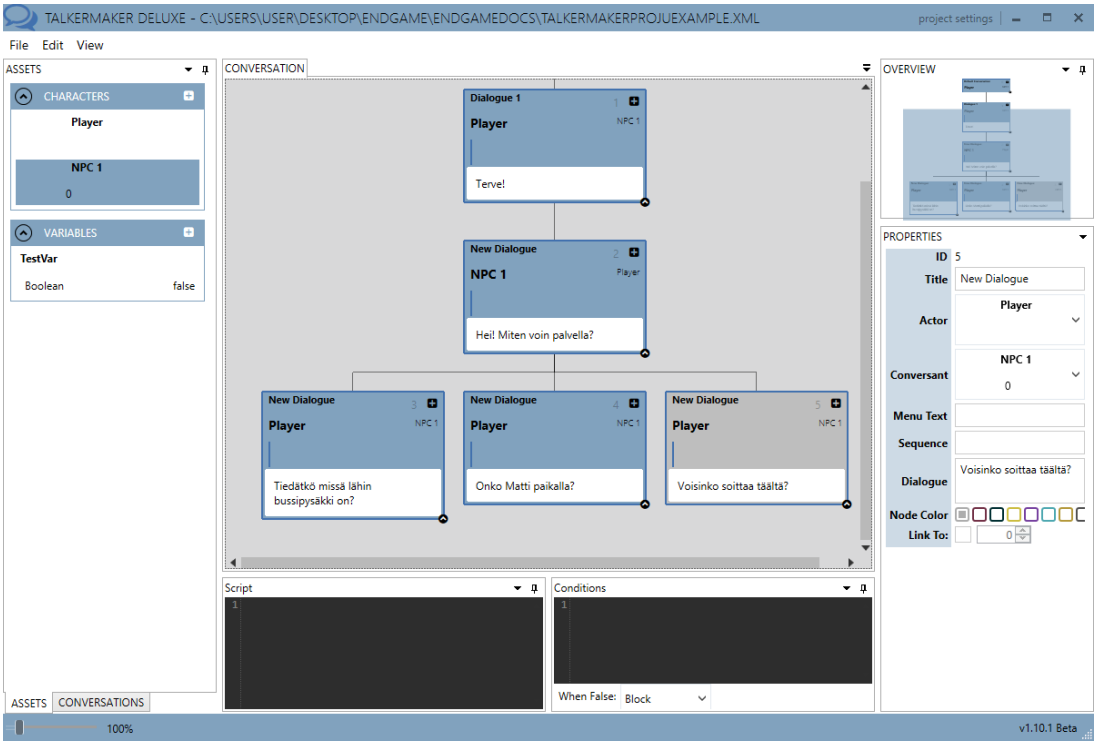
Chat Mapper mahdollistaa kiitettävästi tallennusmuodon asettamisen juuri niin kuin käyttäjä haluaa. Käyttäjä saa päättää, mitkä arvot tallennetaan tiedostoon, ja myös muun muassa, tallennetaanko tyhjiä kenttiä tiedostoon.

Chat Mapper täyttää lähes kaikki tämän insinöörityön dialogieditorilta vaaditut ominaisuudet, mutta tuote vaatii taloudellista investointia käyttäjältä ollakseen todella käyttökelpoinen edes projekteissa, joitten tarkoitus ei ole tuottaa kaupallinen.

Chat Mapperistä on myös raportteja, joiden mukaan keskusteluissa ohjelman sisällä voi olla vain suhteellisen rajallinen määrä solmuja, ennen kuin tietokone hidastuu sietämättömään tasoon, mikä tarkoittaa, että pitkät keskustelut eivät sovi ohjelmalle (23).

2.3.3 TalkerMaker Deluxe – Chat Mapperin kopio

Ratkaistaakseen pitkien keskustelujen ongelmaa Randall Fitzgerald, toinen Barky Seal Gamesin perustajista, ohjelmoi TalkerMakerDeluxen, joka on avoimen lähdekoodin ohjelma ja jota saa käyttää vapaasti ja ilmaiseksi. TalkerMakerDeluxe on kuitenkin vähintäänkin kyseenalainen, sillä ohjelma on pitkälti tarkka kopio Chat Mapperistä käyttöliittymältään ja tallennusformaatiltaan, joka on lähes identtinen Chat Mapperin kanssa. Ohjelman ensimmäinen lataus GitHubiin tehtiin 30.3.2015, joten on mahdollista, että Chat Mapper ei ole vielä tietoinen, että ohjelma on kopioitui näin räikeällä tavalla tai että Chat Mapper ei ole vielä ehtinyt ryhtyä laillisiin toimenpiteisiin (22).



Kuva 12. TalkerMaker Deluxen käyttöliittymä.

Ohjelmassa on hienoisia puutteita tallennusformaatin konfiguroinnissa. Käyttäjä ei pysty esimerkiksi jättämään pois tyhjiä kenttiä tallennuksesta tai päättämään, mitkä kentistä ovat tarpeellisia ja mitkä eivät. Tiedosto ei kuitenkaan tämän seurauksena paisu TalkerMaker Deluxessa kohtuuttomasti, vaan tiedosto pysyy esimerkiksi Articy:draftiin verrattuna hyvin pienenä. Tietynasteista manuaalista karsimista tekstieditorissa tallennuksen jälkeen kuitenkin todennäköisesti tarvitaan.

TalkerMaker Deluxe voi olla varteenotettava vaihtoehto niille, joita ei huoleta ohjelman laillisen tason kyseenalaisuus, tuen puute tai kevyt manuaalinen tekstieditointi, mutta ohjelma voi hyvinkin olla saatavilla vain rajatun ajan.

3 Valinnat ja vaatimukset

3.1 Ohjelmointikielen valinta

C++, C# ja Java ovat ohjelmointikieliä, joita käyttämällä voi luoda Windows-työpöytäsovelluksia suhteellisen helposti ja joista tämän projektin tekijällä on siinä määrin kokemusta, että aikaa voi olettaa menevän enemmän tekemiseen kuin opettelemiseen. Valinta tehdään mainittujen kolmen ohjelmointikielen välillä.

C#:n ja .NET:n käyttö Windows-ohjelmat mielessä tarkoittaa käytännössä joko WinForm-sin (Windows Forms) tai WPF:n (Windows Preservation Foundation) käyttöä. C++:ssa vaihtoehtoina ovat joko C-pohjainen Win32-kirjasto, jo vuodesta 1992 käytetty MFC-kirjasto (Microsoft Foundation Class) tai jokin kolmannen osapuolen kirjasto (24). Javassa on monia vaihtoehtoja, muun muassa AWT (Abstract Window Toolkit), Swing ja SWT (Standard Widget Toolkit).

Microsoft itse sanoo, että C++ on parhaimmillaan, jos halutaan korkein mahdollinen suorituskyky tai tehokkuus, pääsy natiiveihin käyttöjärjestelmäominaisuuksin tai halutaan käyttää DirectX-teknologioita. .NET:n osalta Microsoft mainostaa korkeamman tason koodausta ja suurempaa tuottavuutta. (24.) Käytännössä siis voidaan sanoa, että valinta C++:n ja C#:n välillä on tämän insinöörityön vaatimusten puitteissa valinta suorituskyvyn ja tuottavuuden välillä. Dialogieditori tulee olemaan kevyt ja yksinkertainen ohjelma, joten suorituskykyongelmat ovat epätodennäköisiä, ja näin ollen tuottavuus eli C# on voittaja näiden kahden välillä.

”Jos tähtäimenä on Windows-käyttäjäkunta, olisi melko hankala keksiä syytä käyttää Javaa C#:n sijasta työpöytäkehityksessä”, sanoo John Sonmez, Soft Skills: The Software Developer’s Life Manual -kirjan kirjoittaja (25). Javan suurin etu C#:iin verrattuna lienee sen toimivuus useilla alustoilla, mutta dialogieditori on suunnattu Windows-käyttöjärjestelmille, minkä vuoksi C# ja .NET nimenomaan Windows-ohjelmointiin suunniteltuina ovat mielekäs ja itsestään selvä vaihtoehto. C#:n valinta tarkoittaa myös, että voidaan

käyttää Visual Studiota, joka on IDE:istä se, josta insinöörityön tekijällä on eniten kokemusta.

Microsoftin mukaan WPF on parempi valinta työpöytäsovelluksissa, jos ohjelma vaatii ”monimutkaisia käyttöliittymiä, tyylikustomointia ja graafisesti vaativia skenaarioita” (24). Microsoft vakuuttaa kuitenkin, että vanhempi Windows Forms on yhä hyvä vaihtoehto työpöytäsovelluksille ja että WinForms on kevyempi ja helpompikäyttöisempi vaihtoehto WPF:ään verrattuna. Dialogieditori ei sisällä graafisesti monimutkaisia osioita, ja sen käyttöliittymä tulee olemaan yksinkertainen, joten WPF:n hyödyt jäävät tämän projektin osalta vaatimattomiksi. Tämän lisäksi WinFormsista on insinöörityön tekijällä kokemusta ja WPF:stä ei. Näitten seikkojen perusteella Windows Forms on lopullinen valinta.

3.2 Dataformaatin valinta

3.2.1 Kilpailijat

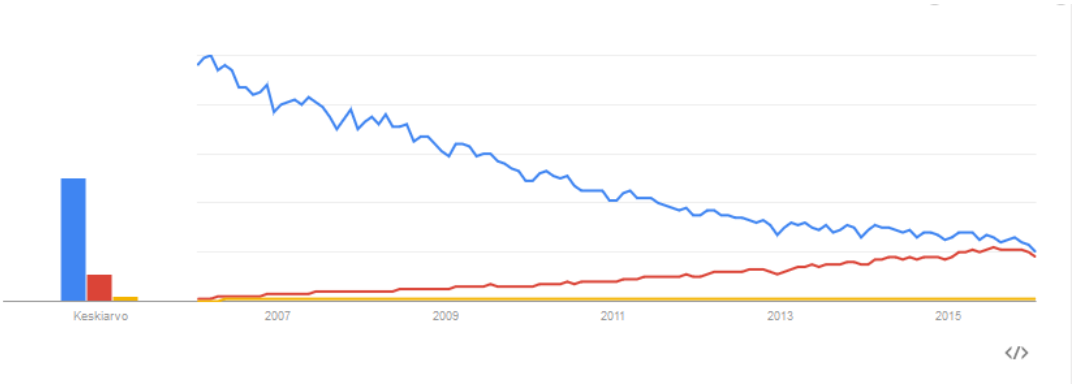
Dataformaatin valinnassa ensisijaisen tärkeänä pidettiin formaatin tarjoamien ominaisuuksien sopivuutta ohjelman vaatimuksiin ja formaatin suosiota sekä oletettavaa elinikää. Datatiedostojen halutaan olla ihmisen luettavissa ilman dialogieditoria, mikä karsii pois binääriformaatit, kuten Apachen Thrift, Googlen Protocol Buffers ja XML-kieleen pohjautuva EXI (Efficient XML Interchange). Näiden formaattien etuna olisivat olleet merkityksellisesti pienemmät tiedostot ja mahdollisesti parantunut suorituskyky (kuva 17), mutta tiedostoja ei pystyisi muuttamaan ilman joko tämän insinöörityön puitteissa tuotettua ohjelmaa tai mittavaa tulkintaprosessia, sillä binääriformaatit eivät ole ihmisen luettavissa.

Myös CSV (Comma Separated Values) -formaatin voi todeta olevan sopimaton tähän ohjelmaan, sillä se sallii vain yhdentyyppisiä ”tallenteita” yhden tiedoston sisällä, ja näillä tallenteilla pitää kaikilla olla yhtä monta kenttää. Toisin sanoen CSV on tarkoitettu taulukkomaisen datan tallennukseen, ja tämä johtaisi lopulta siihen, että dialogieditorin tiedostoista tulisi käytännössä lukukelvottomia ja niiden koko paisuisi.

YAML (YAML Ain’t Markup Language) olisi ominaisuuksiltaan sopiva ohjelmaan, mutta kieli on vähemmän tunnettu ja sen suosio on paljon vaatimattomampi kuin esimerkiksi XML:n tai JSONin. YAML on myös monelta osin samankaltainen kuin JSON. YAML-

kielen virallinen määrittelykin sanoo, että se on laajennettu JSON ja että jokainen JSON-tiedosto on myös pätevä YAML-tiedosto (26). Näiden seikkojen perusteella myös YAML on kieli, jota ei ohjelmaan valittu.

Kielten suosiota tarkasteltaessa käytettiin Google Trendsiä (27) hakusanoilla ”<kieli> tutorial”, eli esimerkiksi ”YAML tutorial”. Tämä on sama menetelmä, jota esimerkiksi tunnettu ohjelmointikielten suosiota mittaava PYPL (PopularitY of Programming Language) käyttää ohjelmointikielten suosiota mitattaessa (28). Kuvassa 13 on Google Trendsin tuottama kaavio XML- (sininen), JSON- (punainen) ja YAML (keltainen) -kielten hakumääräkehityksestä aikaväliltä tammikuu 2006–tammikuu 2016. XML ja JSON ovat jatkuvasti lähestyneet toisiaan ja saavuttaneet miltei tasatilanteen, mutta YAML on tilastollisesti lähes merkityksetön.

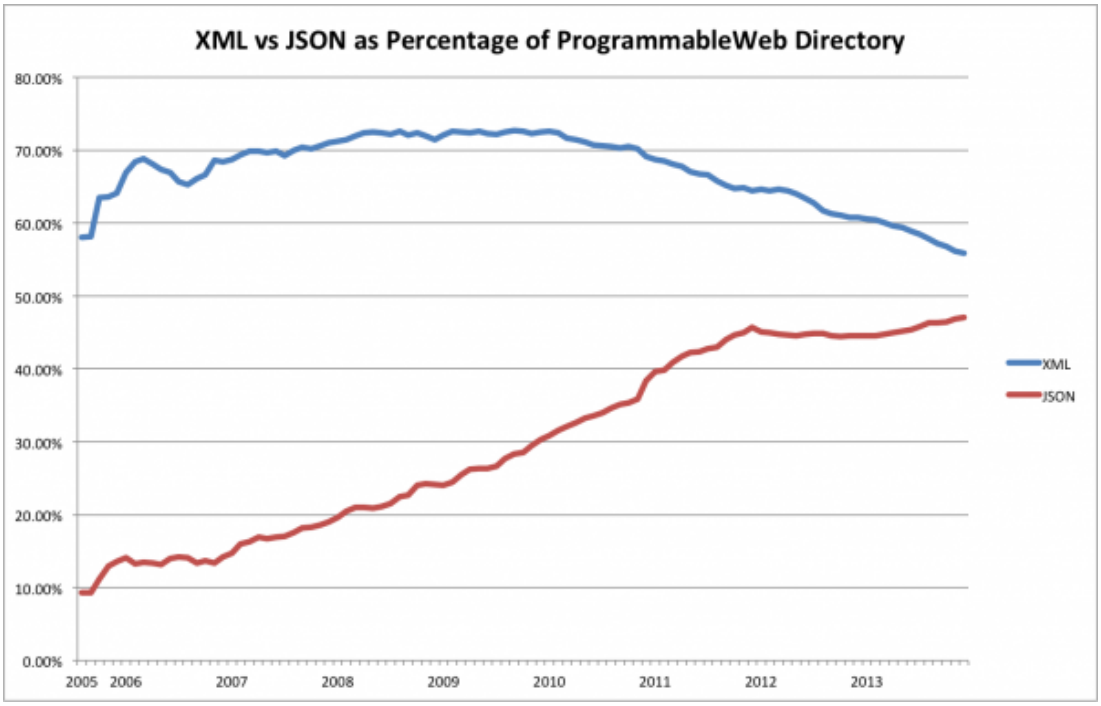


Kuva 13. Google Trends tulokset kielten (XML, JSON ja YAML) suosiota koskien (27).

Toinen käytetty menetelmä on tunnetulla ohjelmointiin liittyvien kysymysten esittämiseen tarkoitetulla sivustolla Stack Overflow (29), ja siellä avainsanoilla tai ”tageilla” löydettyjen tuloksien lukumäärän analysointi. Avainsanoina toimii siis kielen nimi. Kuvassa 14 tuloksien määrät 13.1.2016, ja kuten huomata saattaa, on sekä XML että JSON tuloksia yli 40 kertaa enemmän. XML ja JSON ovat myös Stack Overflow’ssa hyvin lähellä toisiaan suosiossa, joskin JSON on siellä suuremmassa suosiossa.

Kuva 14. Stack Overflow’n hakutuloksien määrät (29).

ProgrammableWeb on verkkosivusto, jolla oli hakemistossaan yli 10 000 ohjelmointirajapintaa vuoden 2013 lopussa (30). Joulukuun 26. päivä 2013 sivusto julkaisi artikkelin, joka vertaili JSON- ja XML-formaattien suosiota sivustolla julkaistuissa ohjelmointirajapinnoissa (kuva 15).



Kuva 15. ProgrammableWeb ohjelmointirajapintahakemiston kasvut XML:n ja JSONin osalta (31).

XML:llä on jälleen hienoinen etu, mutta on selvää, että se on laskusuunnassa ja JSON vastaavasti pitkäjänteisessä nousussa.

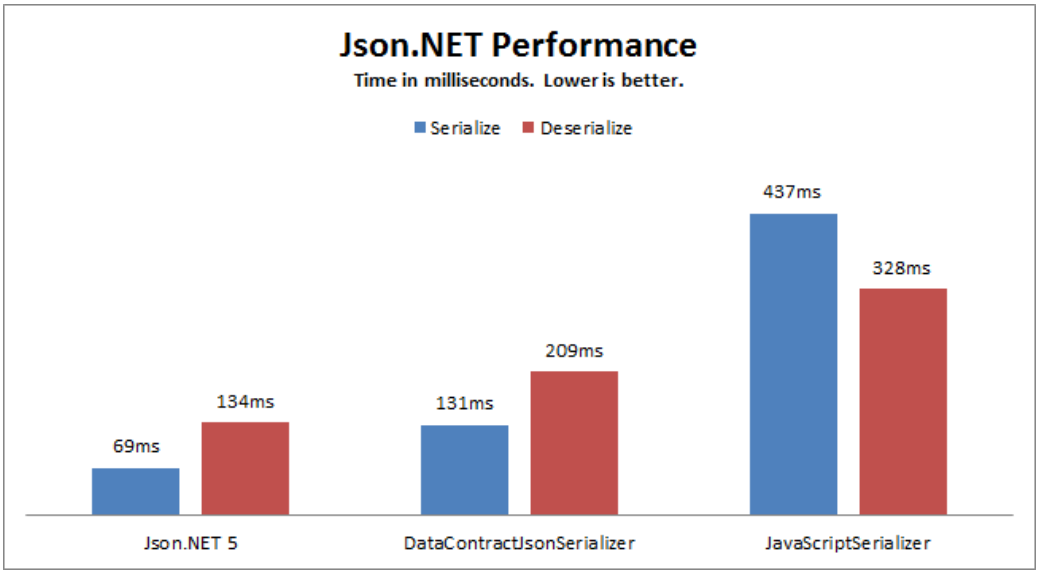
Lopulta dialogieditorissa päädyttiin käyttämään sekä XML:ää että JSONia. Tämä siksi, että formaatit ovat hyvin lähellä toisiaan jokaisessa löydetyssä ajankohtaisessa tilastossa, ja siksi, että vaikka XML onkin vakiintuneempi formaatti ja sillä on laajempi tuki, on vastaavasti JSON hieman nopeampi ja keveämpi. Nämä kaksi formaattia olivat selvästi muita mahdollisuuksia edellä vaatimusten, ominaisuuksien tai yksinkertaisesti suosion vuoksi.

3.2.2 XML ja JSON vastakkain

XML tuki löytyy .NET-kirjastosta suoraan, mikä helpottaa dialogieditorin ohjelmointia, sillä se tarkoittaa, että tarvetta integroida kolmannen osapuolen apukirjastoa ohjelmaan

XML-jäsentämistä varten ei ole. System.Xml-nimiavaruudesta etenkin XmlSerializer tulee olemaan hyödyksi dialogieditoria työstettäessä (32).

JSONia varten .NET tarjoaa muun muassa erinäisiä JavaScript-jäsentimiä, joita voi käyttää JSONin jäsentämiseen, kuten JavaScriptSerializer, ja myös JSON-jäsentimen DataContractJsonSerializer-luokan muodossa, mutta tämä jäsennin on puutteellinen ominaisuuksiltaan ja suorituskyvyltään verrattuna yleisesti käytettyyn Json.NET-jäsentimeen, kuten kuva 16 osoittaa. Json.NET on Newtonsoftin kehittämä, eli se on kolmannen osapuolen jäsennin, mutta se on kuitenkin helppo lisätä projektiin Visual Studiossa viittauslaajennusten kautta.



Kuva 16. Json.NETin suorituskyky vastustajiin verrattuna (33).

On olemassa monia työkaluja XML:n hyödyntämiseen. XPath-kyselykieli mahdollistaa helposti vain halutun tiedon valikoimisen XML-tiedostosta, ja XML-skeemat (muun muassa DTD eli Document Type Definition ja XSD eli XML Schema Definition) ovat hyödyllisiä rakenteen tarkistukseen tarkoitettuja työkaluja tietyissä käyttötarkoituksissa. JSONia varten on myös olemassa vastaavia työkaluja, mutta ne ovat kolmannen osapuolen kehittämiä eivätkä yhtä hyvin integroituja formaattiin kuin XML:ssä, jossa mainitaan skeemat jo määrittelyssä (34). Näistä työkaluista on kuitenkin hyvin rajallisesti hyötyä tämän insinöörityön puitteissa, minkä vuoksi niitä ei työssä käytetty.

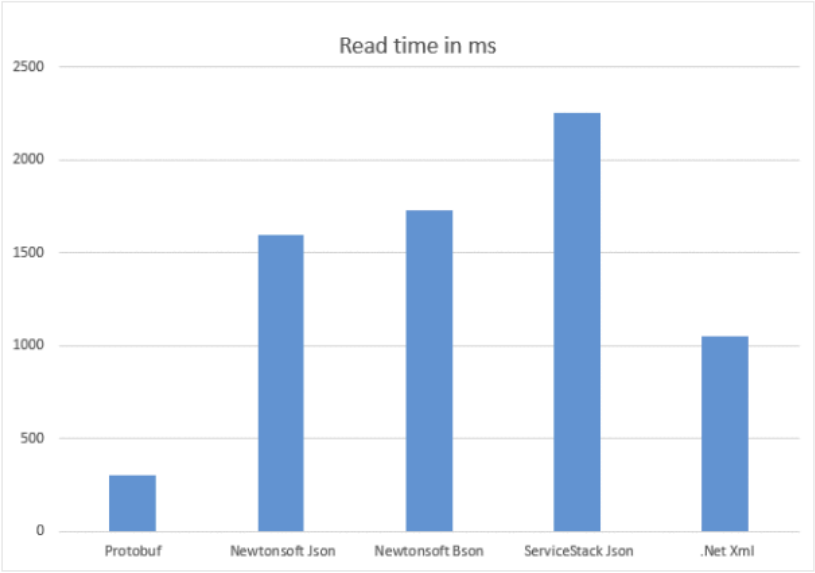
XML:ää moititaan usein sen runsassanaisuudesta. Runsassanaisuus aiheuttaa sen, että XML-tiedostot ovat isoja ja niiden jäsentämisessä kestää kauemmin kuin kilpailijoilla. Yksi virallisen XML-määrittelyn suunnittelutavoitteista onkin: ”Niukkasanaisuudella XML-merkintäkielessä on minimaalinen tärkeys” (34). Formaatin vaikutus tiedostokokoon on

rajallinen dialogieditorin näkökulmasta, sillä hyvin todennäköisesti dialogimateriaali itsessään rajaa tiedostokoon suhteellisen pieneksi.

Guinness World Records -ennätyskirja kuitenkin sanoo, että yksinpeliroolipelien genressä Fallout: New Vegasilla on ennätykselliset 65 000 riviä dialogia (35, s. 147) ja pisimmän käsikirjoituksen titteliä joidenkin lähteiden mukaan taas pitää Planescape Torment 800 000 sanalla (36). Tällaisissa tapauksissa formaatilla on jo merkityksellinen vaikutus, sillä jos oletetaan keskimääräisen sanan koon olevan 20 tavua (37), tarkoittaa 800 000 sanaa jo hieman yli 15 megatavun tiedostokokoa, ja tämä on ennen formaatin pakottamia merkintöjä. Samaan aikaan modernit pelit ovat kuitenkin usein monen kymmenen gigatavun kokoisia (38), ja tässä kontekstissa kymmenet tai sadatkin megatavut ovat pitkälti merkityksettömiä. Formaatin lukemistehokkuus on siis paljon tiedostokokoa merkityksellisempi seikka.

Lukemistehokkuus XML:n ja JSONin paremmuudesta puhuttaessa on kiistanalainen aihe. Aiheesta on useita tutkimuksia kumpaankin suuntaan, mikä tarkoittaa, että paremmuuden toteaminen niiden perusteella on hankalaa.

Kun otetaan huomioon pelkästään tutkimukset, jotka käyttävät täsmälleen tässä työssä potentiaalisesti käytettäviä jäsenninluokkia (Json.NET ja .NET:n Xml-jäsennin), näyttäisi XML nousevan voittajaksi (40; kuva 17).



Kuva 17. Newtonsoft Json ja .NET XML lukemistehokkuus (39).

3.2.3 XML-merkintäkieli

XML eli Extensible Markup Language on yksinkertainen tekstipohjainen formaatti jäsennetyn tiedon kuvaamiseen ja jakamiseen kohteesta toiseen esimerkiksi ohjelmien välillä. XML johdettiin vanhemmasta SGML (Standard Generalized Markup Language) -formaatista internetkäyttöön sopivammaksi (41.) XML-tiedosto ei sisällä tietoa siitä, miten sen sisältö pitäisi näyttää käyttäjille, vaan XML on pelkkää dataa tekstiformaatissa.

Koodiesimerkissä 1 on yksinkertainen esimerkki siitä, miltä XML-dokumentti saattaa näyttää.

```
<muistio>
     <muistutus pvm="12/12/2015">
          <otsikko>Kauppa</otsikko>
          <sisalto>Osta kaupasta vehnäjauhoa</sisalto>
     </muistutus>
     <muistutus pvm="2/1/2016">
          <otsikko>Renkaat</otsikko>
          <sisalto>Muista vaihtaa talvirenkaat autoon
          </sisalto>
     </muistutus>
</muistio>
```

Koodiesimerkki 1. Yksinkertainen XML-tiedosto.

XML-tiedostoissa täytyy aina olla yksi juuri, jonka alle kaikki alahaarat ja niiden alahaarat sijoittuvat. Toisin sanoen, XML-tiedostot noudattavat puumallia. Tässä puumallissa juurta ja sen jokaista alahaaraa kutsutaan elementiksi. Puu alkaa juurielementistä, ja alahaaroja kutsutaan lapsielementeiksi. Jokaisella elementillä voi olla lapsielementtejä. Jos elementeillä on sama isäelementti, eli ne ovat saman elementin lapsielementtejä, niitä kutsutaan sisarelementeiksi.

XML-elementeillä on aina alku- ja loppumerkintä, ja niiden nimet vastaavat toisiaan, mutta mikäli elementin sisällä ei ole tarkoitus olla mitään, voidaan nämä kaksi merkintää yhdistää. Myös kirjainkoon pitää pysyä samana alku- ja loppumerkinnöissä. Esimerkiksi <elementti>-merkinnän täytyy loppua </elementti>-merkintään ja vastaavasti <Elementti>-merkinnän </Elementti>-merkintään. Yhdistetty merkintä olisi esimerkiksi <elementti/>

Elementeillä voi olla jokin tekstipohjainen arvo, ja niillä voi olla alkumerkinnän sisälle istutettuja attribuutteja, jotka ovat aina ainutlaatuisia yhden elementin sisällä, eli toisin

sanoen, yhdellä elementillä voi esiintyä esimerkiksi ”id”-attribuutti vain kerran. Attribuuttien arvot ovat lainaus- tai heittomerkkien sisällä, esimerkiksi <elementti attribuutti=”1”>arvo</elementti>

Tietyt merkit aiheuttavat ongelmia XML:n dataan sijoitettuina, koska niitä ei tulkita puhtaana datana. Nämä merkit ovat <-merkki, >-merkki ja &-merkki. Jäsennin tulkitsee isompi ja pienempi kuin -merkit alku- ja loppumerkintöjen aloitukseksi tai lopetukseksi, ja &-merkki on tarkoitettu merkkiviittausten aloitukseen. Mikäli dataan on tarkoitus sijoittaa joitakin näistä merkeistä, täytyy käyttää merkkiviittauksia hyödyksi. Esimerkiksi pienempi kuin -merkkiviittaus on &lt, ja jäsennin sitten tulkitsee sen pelkäksi <-merkiksi. Attribuuttien kohdalla myös lainaus- ja heittomerkit tarvitsevat merkkiviittausta.

Kommentointi XML:ssä alkaa merkkijonolla <!-- ja loppuu merkkijonolla -->. Kommentoinnilla voidaan mahdollisesti tämän työn yhteydessä selkeyttää tiettyjen merkintöjen merkitystä dialogieditorin ulkopuolista dialogieditointia varten.

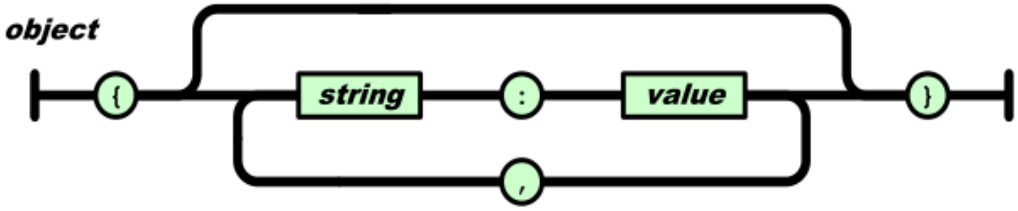
3.2.4 JSON-tiedostomuoto

”JSON (JavaScript Object Notation) on kevyt datanvaihtoformaatti. Ihmisten on helppo lukea ja kirjoittaa ja koneiden on helppo jäsentää ja tuottaa sitä.” Näin sanoo JSONin virallinen sivu vapaasti käännettynä. JSON perustuu Javascriptiin, ja se on muuten riippumaton ohjelmointikielestä, mutta käyttää monia käytäntöjä. (42.)

Kaikki JSON-data on nimi-arvopareissa. Arvot JSONissa voivat olla numeroita (kokonais- tai liukulukuja), merkkijonoja, totuusarvoja, tyhjiä merkkejä, taulukoita tai objekteja (42). Tässä projektissa suurin osa tai kaikki näistä tulevat käytetyksi.

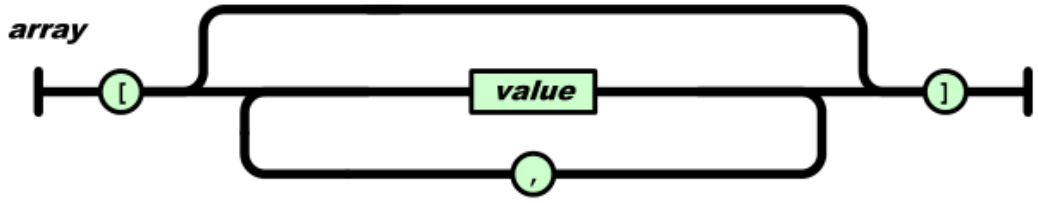
JSONissa on kahdentyyppisiä rakenteita, ja nämä rakenteet ovat kokoelma nimi-arvopareja ja järjestetty lista -arvoja. Toisin sanoen JSONissa kaikki on objektien ja taulukoiden sisällä.

Yksittäisessä nimi-arvoparissa nimi on aina merkkijono, ja se aloitetaan ja lopetetaan lainausmerkillä. Yksittäinen objekti taas sisältää yhden tai useampia nimi-arvoparin pilkulla erotettuna ja aaltosulkeiden sisällä kuvan 18 kaavion mukaisesti. Esimerkiksi henkilö-objekti voisi sisältää etunimen ja sukunimen seuraavanlaisesti: { ”Etunimi”:”Mika”, ”Sukunimi”:”Alasalmi” }.



Kuva 18. JSON-objekti, joka sisältää nimi-arvopareja (42).

Taulukko taas voi sisältää sarjan objekteja hakasulkeiden sisällä ja pilkuilla eroteltuna kuvan 19 ja koodiesimerkki 2:n mukaisesti.



Kuva 19. JSON-taulukko (42).

```
[
{“Etunimi”:”Mika” , ”Sukunimi”:”Alasalmi”},
{“Etunimi”:”Matti” , ”Sukunimi”:”Meikäläinen”},
{“Etunimi”:”John” , ”Sukunimi”:”Doe”}
]
```

Koodiesimerkki 2. JSON-taulukko.

Se, että JSON-kielessä ei käytetä alku- ja loppumerkintöjä, hankaloittaa sarjallistamista hiukan, sillä siinä missä XML:n merkinnät takaavat aina, että objektien tyyppi on näissä merkinnöissä, ei JSON takaa tätä mitenkään. Käytännössä se tarkoittaa, että objektien mukaan pitää lisätä ylimääräinen arvo takaamaan lukemiskyvyn.

3.3 Vaatimusmäärittely

Vaatimuksien tunniste (ID) riippuu kategoriasta. Toiminnalliset vaatimukset alkavat TV:llä, ja numero kasvaa järjestyksessä. Esimerkiksi viides toiminnallinen vaatimus on siis TV05. Ei-toiminnallisissa vaatimuksissa kirjainyhdistelmä ETV aloittaa tunnisteen, mutta tunnisteen lisänä on erillinen tarkentavaa kategoriaa kuvaileva kirjain tai kirjaimet. Esimerkiksi toimintavarmuuskategoriasta ei-toiminnallisissa vaatimuksissa tunniste voi olla ETVTV02.

Prioriteetti esitetään vaatimuksissa numerolla 1–3, jossa 1 = pakollinen, 2 = tarpeellinen ja 3 = toivottava.

Toiminnalliset vaatimukset määrittelevät toimintoja, joita järjestelmän tai järjestelmän komponentin pitää pystyä suorittamaan (43). Taulukko 1:ssä on listattu dialogieditorin toiminnalliset vaatimukset.

Taulukko 1. Toiminnalliset vaatimukset

ID	Priori-teetti	Kuvaus
TV01	1	Ohjelmalla pystytään tuottamaan datana dialogipuuformaatissa näkyvää dialogia.
TV02	1	Dialogipuun pystyy tallentamaan JSON-formaatissa.
TV03	2	Dialogipuun pystyy tallentamaan XML-formaatissa.
TV04	1	Ohjelman tuottamat tiedostot noudattavat valitun tallennusformaatin syntaksisääntöjä.
TV05	2	Ohjelman tuottamat tiedostot ovat rivitettyjä ja sisennettyjä luettavuuden helpottamiseksi.
TV06	2	Ohjelmassa voi olla auki useampia kuin yksi dialogipuu samanaikaisesti.
TV07	3	Avattaessa ohjelma aukaisee suljettaessa auki olleet tiedostot.
TV08	1	Dialogipuuhun voi sijoittaa NPC:eitä.
TV09	1	NPC:eiden alle voi sijoittaa NPC-repliikkejä.
TV10	1	NPC:t sisältävät tiedon siitä, mikä niiden ensimmäinen NPC-repliikki on.

TV11	2	Ensimmäinen NPC-repliikki voi riippua ehdoista. Toisin sanoen ensimmäisiä NPC-repliikkilinkkejä voi olla useita, ja data sisältää tiedon kaikesta, mitä tarvitaan päätöksen tekemiseen.
TV12	3	NPC:n voi valita olevan geneerinen, jolloin se sisältää tiedon vain ensimmäiseen NPC-repliikkiin johtavista asioista.
TV13	1	Pelaajan vastausvaihtoehtoja voi olla useita.
TV14	2	Pelaajan vastausvaihtoehdoilla voi olla yksi tai useampia ehtoja niiden aktivoimiseksi.
TV15	1	Pelaajien vastausvaihtoehdot voivat johtaa NPC-repliikkeihin missä tahansa kohtaa dialogipuuta.
TV16	2	NPC:eiden ja pelaajien repliikit voivat johtaa myös erinäisiin toimintoihin, esimerkiksi äänen toistamiseen tai jonkin arvon muutokseen.
TV17	2	Käyttäjän tulee voida räätälöidä ehtoja ja toimintoja.
TV18	3	Pelaajan vastausvaihto voi sisältää tiedon ennen valintaa pelaajalle näkyvästä arvosta ja erikseen siitä, mitä pelaajan hahmo oikeasti sanoo. Esim. pelaajalle vaihtoehtona annetaan ”Tervehdi”, mutta pelaajan hahmo sanoo ”Hei Matti. Mitä kuuluu?”
TV19	3	Pelaajalla voi pelissä olla hallussaan useampia kuin yksi hahmo, ja pelaajan repliikki voi kuulua kenelle tahansa heistä.
TV20	3	Ohjelman oletussolmujen (NPC, NPC-repliikki ja pelaajan vastausvaihtoehto) lisäksi ohjelman tulee tukea käyttäjän räätälöimiä solmuja.
TV21	3	Koska räätälöityjen solmujen tarkoitusta tai tyyppiä ei voi tietää, tulee niitä pystyä sijoittamaan minkä tahansa solmun alle ja minkä tahansa muun solmun voi sijoittaa niiden alle.
TV22	1	Luotuja solmuja voi poistaa.
TV23	2	Mikäli poistettava solmu sisältää lapsisolmuja, ohjelma tiedottaa käyttäjälle tästä ja varmistaa tämän aikeen.
TV24	2	Solmun alla ei voi olla samaa tyyppiä olevaa solmua.
TV25	3	Ohjelma voi tuottaa näkymän dialogin kulusta. Tässä näkymässä näkee jokaisen keskusteluhaaran kulun kokonaisuudessaan.
TV26	3	Ohjelma voi tuottaa käsikirjoitusmallisen dokumentin dialogista.
TV27	2	Solmuja voi kopioida, leikata ja liimata.
TV28	2	Solmujen nimet vaihtuvat niiden arvon vaihtuessa tunnistamisen helpottamiseksi.

TV29	1	Valitun solmun attribuutit näkyvät ja ovat muokattavissa. Myös sen sisäiset mahdolliset ehdot ja toiminnot näkyvät ja ovat muokattavissa valittaessa.
TV30	3	Solmuja voidaan etsiä arvon perusteella.
TV31	3	Ohjelman sisällä voidaan testata dialogia.

Taulukossa 2 on listattu käytettävyysvaatimukset, jotka kuuluvat ei-toiminnallisiin vaatimuksiin. Käytettävyys tarkoittaa sitä, miten helposti käyttäjä oppii käyttämään ohjelmaa, valmistelee syötteet ja tulkitsee ohjelman tuottamia tuloksia (43).

Taulukko 2. Käytettävyysvaatimukset

ETVK01	2	Ohjelman tulee olla käyttöliittymältään välittömästi tutun oloinen Windows-ympäristöön ja -ohjelmiin tottuneille.
ETVK02	1	Ohjelman ikkunan kokoa voidaan muuttaa, ja käyttöliittymä mukautuu kokoon.
ETVK03	2	Oikealla hiiren napilla tehtävät asiat voidaan tehdä myös vaihtoehtoisella käyttöliittymänapin painalluksella.
ETVK04	2	Jos tietyt osat käyttöliittymää eivät mahdu senhetkiseen ikkunaskaalaan, nämä osat ottavat automaattisesti vierityspalkin käyttöön.
ETVK05	2	Painikkeet, joita ei voi käyttää tietyllä hetkellä, ovat silloin selkeästi merkittyjä (esim. harmaita).

Taulukko 3:ssa on listattu sekä toimintavarmuus- että suorituskykyvaatimukset. Molemmat vaatimuskategoriat kuuluvat ei-toiminnallisiin vaatimuksiin.

Taulukko 3. Toimintavarmuus- ja suorituskykyvaatimukset

ETVTV01	2	Jos ohjelmassa avataan väärän formaatin XML- tai JSON-tiedosto, ohjelma ei kaadu, vaan ilmoittaa käyttäjälle tästä virheestä.
ETVTV02	2	Väärä käyttäjäsyöte arvoja muokattaessa ei kaada ohjelmaa.
ETVS01	1	Mikään muu kuin tallennus ja lataus eivät saa aiheuttaa suurempaa hidastumista kuin ihmisen keskimääräinen reaktioaika (n. 0,2 sekuntia).

ETVS02	2	Lataus ja tallennus saavat aiheuttaa merkityksellisiä hidastuksia, mutta ohjelman pitää pystyä lataamaan projekti, joka sisältää 2000 solmua alle sekunnissa (rajana pidettäisiin tällöin dialogipuuta, jossa olisi 10 henkilöä, josta jokaiseen liittyy 200 solmua).

4 Toteutus

4.1 Elementit

4.1.1 Datatiedoston elementit

Datatiedoston muotoa kehiteltäessä on luonnollista aloittaa siitä, mitä elementtejä datatiedosto itse asiassa pitää sisällään.

Pohjalta aloittaen **dialogivaihtoehto** eli pelaajan vastausvaihtoehto on elementti, joita voi olla monta yhdessä solmussa vaatimuksen TV13 mukaisesti. Toisin sanoen, dialogivaihtoehdolla voi olla monta sisarelementtiä. TV14- ja TV16-vaatimusten mukaisesti dialogivaihtoehdoilla voi olla myös ehtoja ja toimintoja.

Dialogivaihtoehtojen olemassaolo ei ole loogista riippumattomana elementtinä, vaan niillä on tarkoitus ainoastaan toisten elementtien alla. Tämän lisäksi vaihtoehtojen sisaruus on mahdollista vain saman elementin alapuolella. Yksi mahdollisuus äitielementiksi on **NPC-repliikki**, sillä keskustelut peleissä käytännössä aina alkavat NPC:eiden repliikeillä pelaajan sijasta. NPC-repliikki voi myös palvella kaksoistarkoitusta, jos käyttäjälle annetaan mahdollisuus käyttää sitä myös tyhjänä dialogisolmuna, eli solmuna, jonka tarkoitus on olla vain dialogivaihtoehtoja yhdistävä tekijä. TV16-vaatimuksen mukaisesti NPC-repliikillä voi myös olla toimintoja.

Vaatimuksen TV15 mukaisesti dialogivaihtoehdot voivat johtaa NPC-repliikkeihin missä tahansa kohtaa dialogipuuta. Käytännössä tämä tarkoittaa, että kaikkien NPC-repliikkien tulee olla mistä tahansa saatavilla. Juureen sijoittaminen on mahdollista, mutta tiedoston lukeminen ilman editoria olisi lähes sietämätöntä, joten niiden sijoittaminen alimpaan mahdolliseen NPC-repliikkejä yhdistävään elementtiin on järkevää.

Räätälöity solmu ja **NPC** itse ovat NPC-repliikkejä mahdollisesti yhdistävät elementit. NPC on elementti, joka helpottaa editointia ja todennäköisesti jouduttaa myös tiedoston lukemista pelissä, sillä sen käyttö tarkoittaa tietyistä kilpailevista tuotteista poiketen sitä, ettei käyttäjän tarvitse jatkuvasti toistaa, ketkä ovatkaan puhumassa tällä kertaa. NPC-repliikin ei siis toisin sanoen tarvitse sisältää tietoa puhujasta, koska sen äitielementti kertoo tämän käyttäjälle. Tästä voi kuitenkin koitua ongelmatilanteita, jos käyttäjä haluaa, että keskustelussa on mukana muitakin kuin yksi NPC ja pelaaja. NPC-elementin täytyy pitää sisällään TV10- ja TV11-vaatimusten mukaisesti tieto ensimmäisestä NPC-repliikistä, ja tämä repliikki voi riippua ehdoista.

Räätälöity solmu on editorin yksinkertaisin elementti, sillä se sisältää vain kaksi merkkijonomuuttujaa (solmun tyyppi ja arvo). Sen tarkoitus on lähinnä yhdistää dialogielementtejä saman katon alle geneerisyyden mahdollistamiseksi tai organisoinnin vuoksi. Solmun tyyppi voi olla esimerkiksi ”Paikka” ja arvo ”Helsinki”, jolloin elementin alle sijoittuisi kaikki Helsingissä tapahtuva dialogi. Tällä tavalla voidaan mahdollistaa muun muassa esimerkin mukainen paikkakohtainen, mutta hahmosta riippumaton dialogi.

XML tarvitsee aina yksittäisen **juurielementin**, minkä vuoksi myös tämän editorin on pakko käyttää sitä kaikkien äitinä.

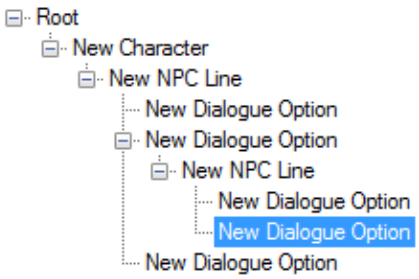
Koodiesimerkki 3:ssa on yksinkertainen esimerkki siitä, miltä data saattaa näyttää XML-tiedostona.

```
<?xml version="1.0"?>
<Root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Children>
    <DialogueTreeNode xsi:type="Character">
      <Children>
        <DialogueTreeNode xsi:type="NPCLine">
          <Children>
            <DialogueTreeNode xsi:type="DialogueOption">
              <DialogueValue>Hei Matti!</DialogueValue>
            </DialogueTreeNode>
            <DialogueTreeNode xsi:type="DialogueOption">
              <DialogueValue>...</DialogueValue>
            </DialogueTreeNode>
            <DialogueTreeNode xsi:type="DialogueOption">
              <DialogueValue>Hyvästi</DialogueValue>
              <NextNodeRef>1</NextNodeRef>
            </DialogueTreeNode>
          </Children>
          <DialogueValue>Hei!</DialogueValue>
          <ID>0</ID>
        </DialogueTreeNode>
           <DialogueTreeNode xsi:type="NPCLine">
          <DialogueValue>?!</DialogueValue>
          <ID>1</ID>
        </DialogueTreeNode>
      </Children>
      <Name>Matti</Name>
    </DialogueTreeNode>
  </Children>
</Root>
```

Koodiesimerkki 3. Esimerkki pienestä dialogidatatiedostosta.

4.1.2 Elementit käyttöliittymässä

Windows Formsin tarjoama TreeView-luokka on lähes ihanteellinen sekä ominaisuuksiltaan että käytännöllisyydeltään dialogipuun vaatimuksille. Ilman erillisiä tekijän toimenpiteitä TreeView tarjoaa muun muassa eri solmujen pienentämisen ja laajentamisen sekä selvät sisennykset eri elementtien välillä (kuva 20).



Kuva 20. TreeView-luokka käytössä.

Selkeyden voi sanoa kuitenkin olevan dialogin kulun kannalta hiukan heikompi kuin vuokaaviomaisten kilpailijoiden ratkaisujen, sillä TreeView näyttää samalla hetkellä huomattavan paljon enemmän yksittäisiä solmuja, ja niiden erittely yhdellä silmäyksellä on mutkikkaampaa. Vuokaavioformaatti tarkoittaa kuitenkin laajoissa dialogipuissa huomattavaa määrää vierittelyä edestakaisin, sillä yksittäiset elementit vievät niissä paljon enemmän tilaa. Huomionarvioista on myös, että vuokaaviossa miltei kaikki yksityiskohdat ovat välittömästi näkyvissä, siinä missä TreeView-vaihtoehdossa näkyy lähinnä keskeisin tai korkeintaan muutama arvo yksittäisestä elementistä.

Käytännössä valinta siis riippuu siitä, mitkä asiat ovat prioriteetteja käyttöliittymässä. TreeView tarjoaa nopeutta ja ylätason perspektiiviä ja vuokaavio taas selkeää dialogin kulkua ja välitöntä yksityiskohtien näkemistä. Tässä dialogieditorissa nopeus on selkeyden edellä prioriteettina, joten TreeView vie voiton näiden kahden vaihtoehdon välillä.

Käyttöliittymän dialogipuun ja lopullisen datatiedoston välillä on kuitenkin merkittävä ero, joka tuottaa ongelman editorin kannalta: käyttöliittymän puussa NPC-repliikit voivat olla syvällä esimerkiksi pelaajan dialogivaihtoehtojen alla, mutta jokaisen NPC-repliikin pitää olla saatavilla mistä tahansa linkitysten vuoksi (TV15). Asia on näin, koska dialogin kulkua olisi erittäin hankalaa tai mahdotonta hahmottaa puusta, jos kaikki NPC-repliikit olisivat samalla tasolla. Käyttöliittymässä voi siis vaikuttaa siltä, että NPC-repliikki x on dialogivaihtoehto y:n alla, mutta datatiedostossa repliikki ei ole y:n alla vaan esimerkiksi hahmon z alla ylempänä puussa. Ongelma ratkeaa, kun erotellaan projektin tallennus

datatiedoston lopullisesta tallennuksesta peliä varten, jolloin projektin tallennuksessa tallennetaan TreeView'n mukaisilla sisennyksillä ja datatiedoston tallennuksessa NPC-repliikit nostetaan oikeaan solmuun.

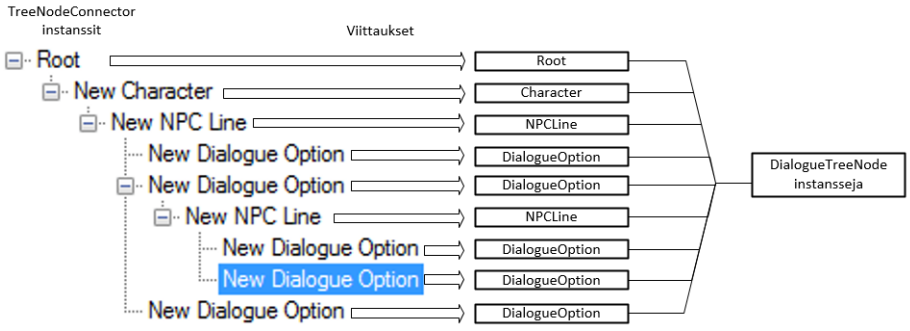
4.1.3 Elementit ohjelmoinnin kannalta

XML- ja JSON-tiedostojen lukemisen ja kirjoittamisen voidaan sanoa olevan ratkaistu. Dialogieditorin kannalta pitää kuitenkin olla tarkkana ohjelman rakenteen kanssa, jottei joudu kehittämään turhaa uutta ratkaisua. Siksi avainsanana ja monien päätöksien takana on sarjallistaminen.

Microsoft määrittelee sarjallistamisen (serialization) näin: ”Sarjallistaminen on prosessi, jossa objekti muunnetaan bittivirraksi sen säilyttämiseksi tai siirtämiseksi muistiin, tietokantaan tai tiedostoon. Sen päätarkoitus on tallentaa objektin tila, jotta se voidaan luoda tarvittaessa uudelleen” (44).

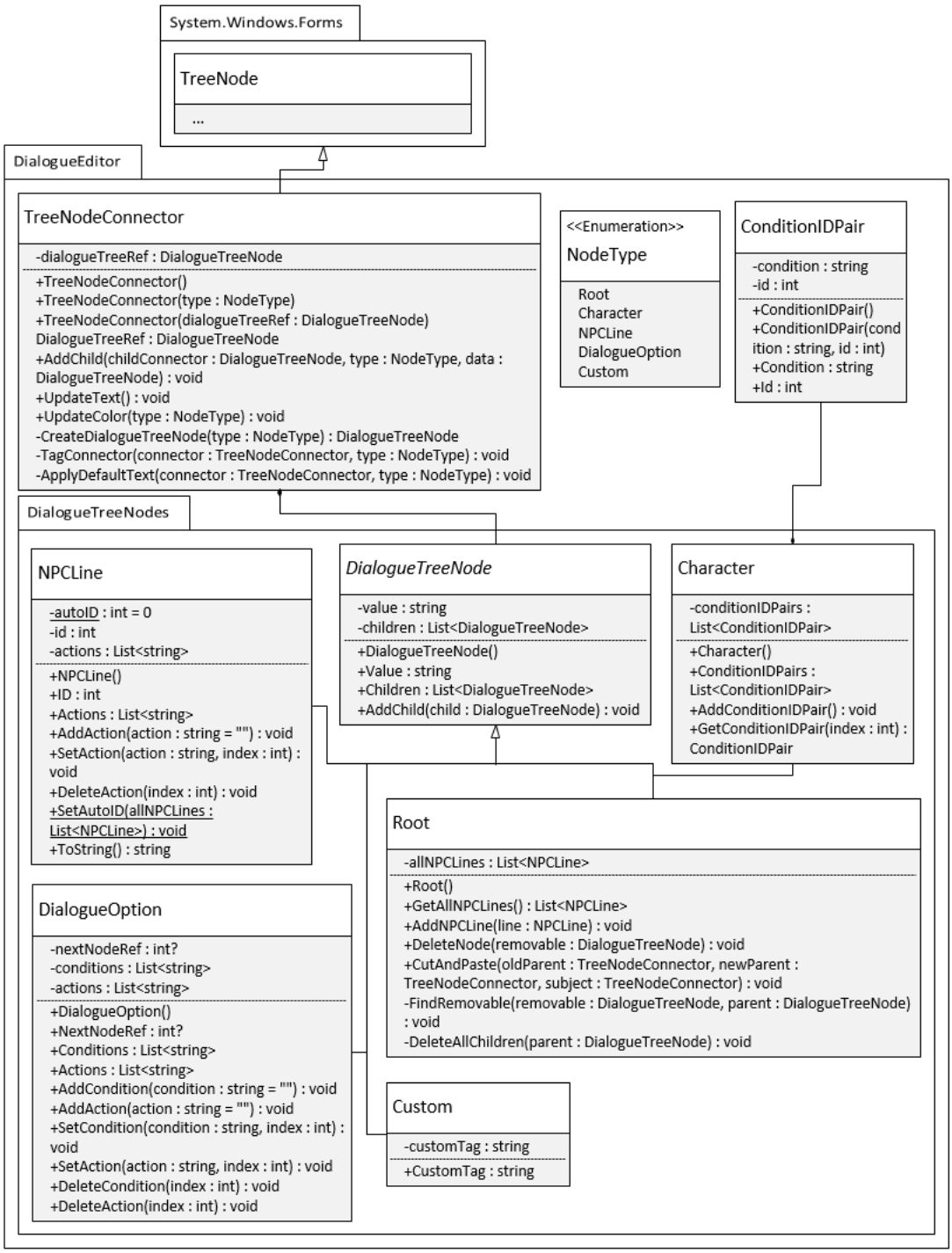
TreeView'ta ja sen solmuluokka TreeNodea ei ole suunniteltu suoraan sarjallistamiseen, joten dialogipuun elementtien kytkeminen näihin luokkiin on huono ajatus. Tätä varten dialogieditoriin luotiin luokka TreeNodeConnector, joka perii TreeNode-luokan ja jonka ainoa uusi kenttä on viittaus dialogipuuelementtiin. Kaikki TreeView-solmut ovat dialogieditorin sisällä instansseja tästä TreeNodeConnector-luokasta.

Kaikki dialogipuuelementit perivät abstraktista DialogueTreeNode-luokasta, toisesta ohjelmaa varten luodusta luokasta, minkä vuoksi TreeNodeConnector yksinkertaisesti sisältää viittauksen instanssiin tästä yläluokasta, kuten kuva 21 osoittaa. Kaikki dialogidata on DialogueTreeNodesta perivissä luokissa.



Kuva 21. TreeNodeConnectorin toiminta.

Kuvan 22 luokkakaaviossa on sekä tämä DialogueTreeNodeConnector että kaikki dialogipuuelementit nähtävissä.



Kuva 22. TreeNodeConnector, DialogueTreeNode ja DialogueTreeNoden lapset UML-luokkakaaviossa.

ConditionIDPair on luokka, jota käytetään hahmon avausrepliikkiä päätettäessä. Käytännössä hahmolle voidaan siis määrittää ehtoa vastaava repliikki, mutta mikäli yhtään tällaista ehto-repliikkiparia ei ole, voidaan päättää esimerkiksi, että hahmon ensimmäinen repliikki aloittaa.

NPCLine eli NPC-repliikki sisältää staattisen muuttujan autoID ja staattisen metodin SetAutoID, joitten avulla uudelle NPC-repliikille annetaan automaattisesti ensimmäinen vapaa kokonaisluku tunnisteeksi. Tunnistetta voidaan muuttaa, mutta ohjelma estää käyttäjää käyttämästä jo olemassa olevaa tunnistetta.

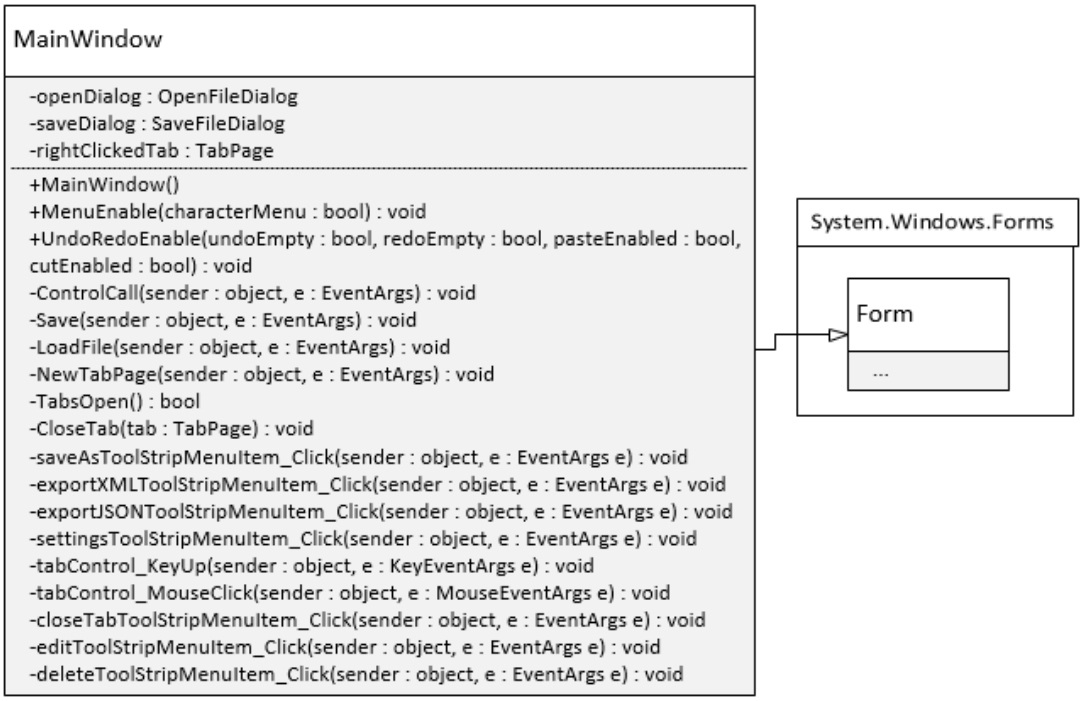
Viittaus NPC-repliikkiin pelaajan vastausvaihtoehdossa on kokonaisluku, joka voi olla ”null”, koska tällä tavoin voidaan helposti tarkistaa, onko arvo olemassa sarjallistamisen yhteydessä. Tällaiset tarkistukset tehdään ShouldSerializeX()-metodeilla, joissa X:n tilalla on tarkistettavan arvon nimi ja jotka palauttavat totuusarvon. Näitä metodeja ei luokkakaaviossa näy, sillä tilaa oli rajatusti.

NPC-repliikin tunniste on vastaavasti myös kokonaisluku. Tunnistetyypin valinta ei ollut itsestään selvää, sillä esimerkiksi merkkijono olisi tarjonnut suuremman määrän vaihtoehtoisia arvoja ja sitä myöten suurempaa mahdollista määrää repliikkitunnisteita. C#:ssa kokonaisluku eli int on kuitenkin 32-bittinen, mikä tarkoittaa, että mahdollisia arvoja kokonaisluvulle on yli neljä miljardia. Ohjelma ei rajoita solmujen määrää, mutta neljän miljardin repliikin saavuttamista voidaan pitää epätodennäköisenä.

4.2 Käyttöliittymä

Ylimmän tason käyttöliittymäelementti dialogieditorissa on MainWindow, Windows Formsin Forms-luokasta perivä luokka. Se sisältää ylävalikot ja välilehtikontrollin, ja se hallitsee myös tiedoston avaus- ja tallennusikkunoita.

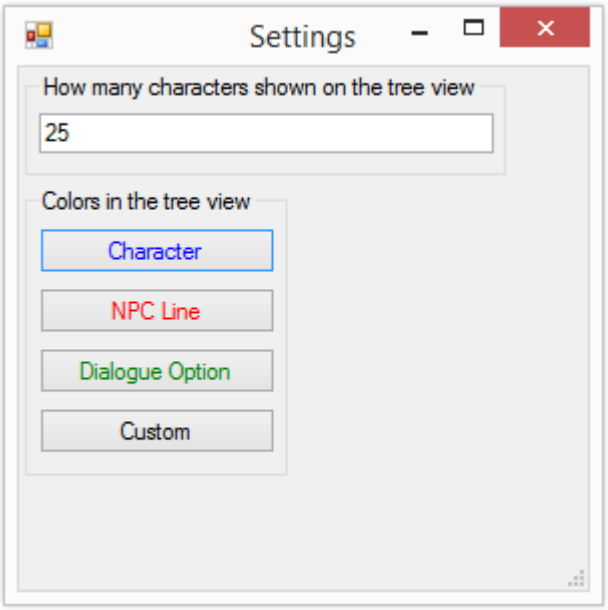
Kuvassa 23 on MainWindow-luokan UML-luokkakaavio. OpenFileDialog ja SaveFileDialog ovat System.Windows.Forms-nimiavaruuden sisäisiä luokkia. Suurin osa MainWindow'n metodeista ovat suoraan käyttöliittymän painikkeista kutsuttuja.



Kuva 23. MainWindow UML-luokkakaavio.

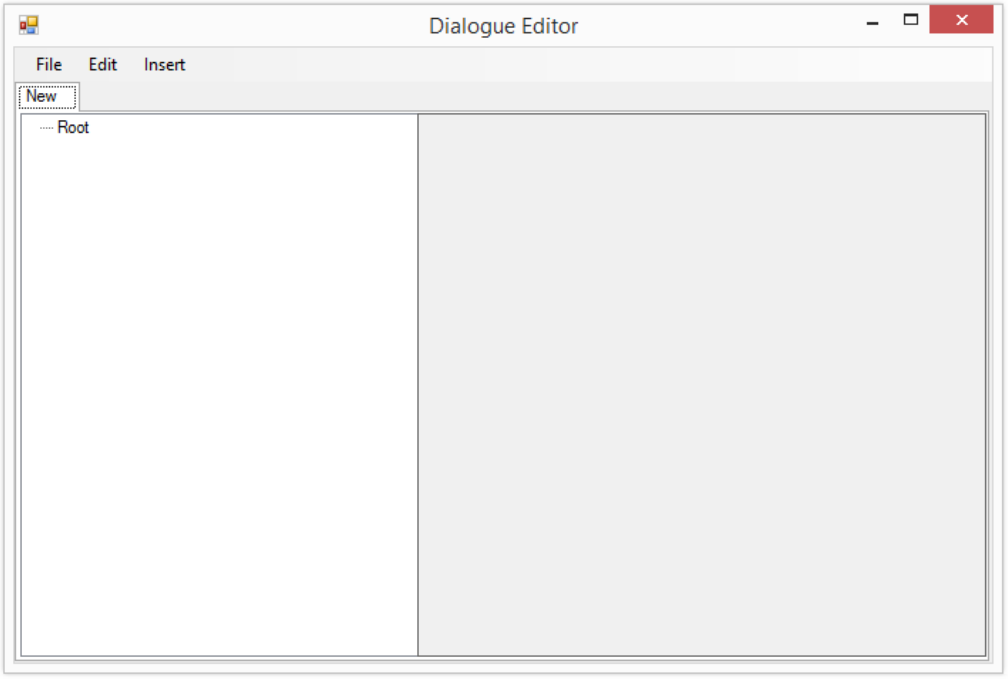
Ylävalikossa avautuu alavalikkoja jokaisesta kolmesta vaihtoehdosta (File, Edit ja Insert). File-valikon sisällä vaihtoehtoina ovat uuden dialogipuun luonti, mikä samalla aukaisee uuden välilehden, dialogipuun tallennus, dialogipuun tallennus nimellä, dialogipuun lataus, dialogipuun vienti XML-tiedostoksi ja dialogipuun vienti JSON-tiedostoksi. Edit-valikko sisältää perustoimintoja, kuten solmun poisto, leikkaus ja liimaus, mutta se sisältää myös ohjelman asetuksiin johtavan painikkeen. Insertissä ovat luonnollisesti elementtien lisäyspainikkeet sillä hetkellä valitun elementin alle. Solmut, joita ei voi lisätä valitun elementi alle, ovat estettyjä ja harmaina.

Kuvassa 24 on ohjelman asetuksia muokkaava ikkuna SettingsEdit. Asetukset tallennetaan .NET-kirjaston tarjoamiin Properties.Settings-asetuksiin, jotka säilyvät ohjelman sammumisen jälkeenkin ja jotka tarjoavat helpon lataamisen ja muokaamisen ohjelman sisällä. Asetuksista päätetään yksinkertaisesti eri solmuluokkien värikoodaus ja kuinka pitkiä solmujen tunnisteet TreeView-näkymässä ovat.



Kuva 24. SettingsEdit-ikkuna.

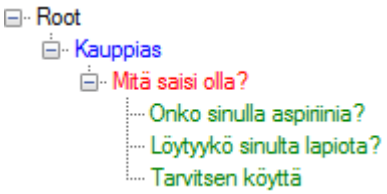
MainWindow'n välilehtikontrollin sisälle asettuvat MainControls-luokan intanssit. MainControls perii System.Windows.Forms-nimiavaruuden UserControl-luokan. MainControls jakautuu kahteen paneeliin. Nämä kaksi paneelia pitävät hallussaan selkeästi suurinta osaa ohjelman ikkunasta (kuva 25).



Kuva 25. MainWindow ja sen sisällä oleva MainControls ohjelman käynnistyksen jälkeen.

Vasemmanpuoleinen paneeli sisältää TreeView-näkymän, mitä voisi pitää ohjelman keskeisimpänä käyttöliittymäelementtinä, sillä se näyttää kaikki dialogipuun solmut ja sen kautta valitaan muokattava solmu. (Kuva 26.)

TreeView'n solmut värikoodautuvat automaattisesti tyypin mukaan (kuva 26). Tämä auttaa erottamaan eri elementit toisistaan lyhyellä vilkaisulla ilman erityisen analyysin tarvetta. Elementeillä on oletusvärit, mutta niitä voidaan muokata kuvan 24 havainnollistamissa asetuksista.

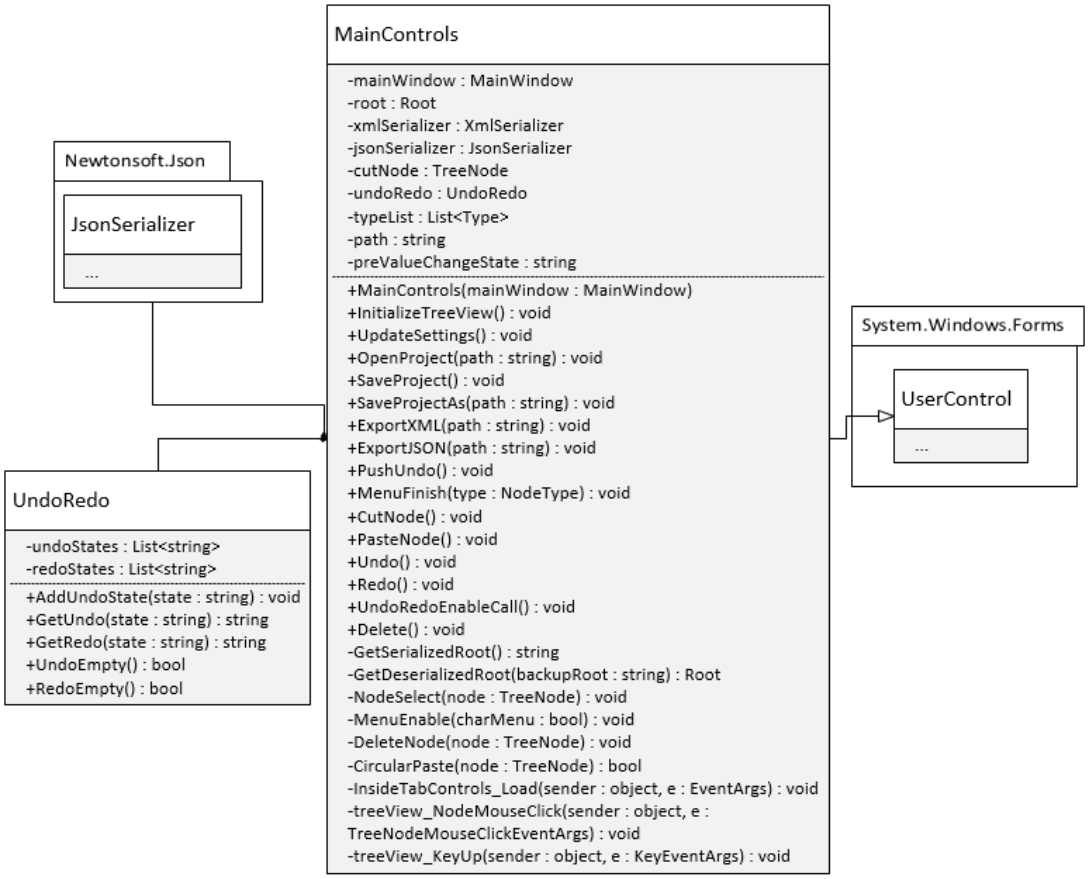


Kuva 26. TreeView ja värikoodatut solmut.

Solmujen arvot TreeView'n sisällä saavat tekstiarvokseen elementtien pääarvot, eli toisin sanoen esimerkiksi kuvan 26 NPC-elementtiä edustava sininen solmu on tekstiarvoltaan "Kauppias," koska NPC-elementin hahmonimi on "Kauppias." Värikoodauksen ja tekstiarvon ansiosta jokainen solmu on pitkälti välittömästi tunnistettava TreeView'n sisällä.

Oikeanpuoleinen paneeli on tyhjä, kun yhtään solmua ei ole valittuna, mutta kun solmu valitaan, paneeli mahdollistaa solmun arvojen muuttamisen. Tämä paneeli on eri riippuen solmun tyypistä.

Kuvassa 27 on MainControls-luokan ja sen sisältämän UndoRedo-luokan UML-luokkakaaviot. TreeView-näkymässä solmua oikealla hiiren näppäimellä klikattaessa saadaan esille valikko, josta voidaan lisätä uusi solmu valitun solmun alle. Valikko on käytännössä nopeampi vaihtoehto Insert-valikolle, ja niiden toiminta on täsmälleen sama. Jokainen valikon vaihtoehdoista käyttää omaa MainControls-metodia, mutta kuvan 27 MainControls-kaaviossa ei ole lueteltu näitä neljää metodia niiden samankaltaisuuden ja tilanpuutteen vuoksi. Esimerkiksi uutta hahmoa lisättäessä käytetään NewCharMenu_Click(object sender, EventArgs e)-metodia, ja loput kolme noudattavat samaa kaavaa.

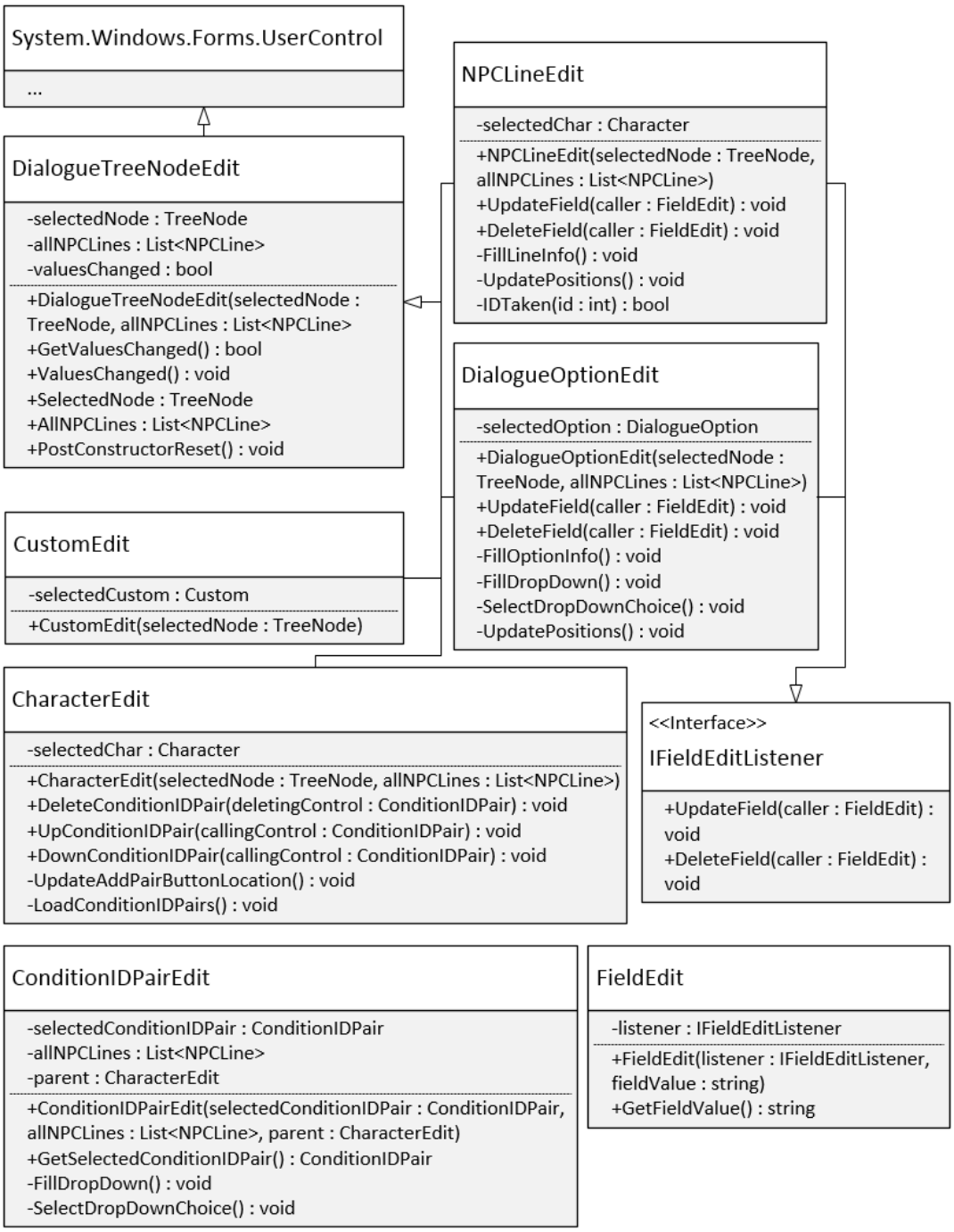


Kuva 27. MainControls-luokan UML-luokkakaavio.

MainControls-luokalla on UndoRedo-luokan instanssi. UndoRedo on nimensä mukaisesti kumoa ja tee uudelleen -operaatioista huolehtiva luokka. Tämä on toteutettu yksinkertaisesti säilyttämällä luokan sisällä dialogipuun tiloja ennen erinäisiä operaatioita. Yk-

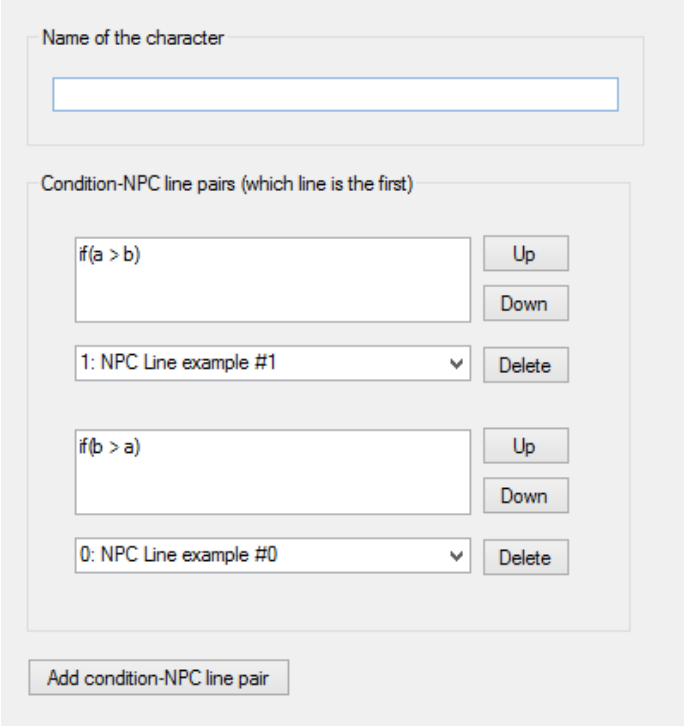
sittäistä tilaa edustaa merkkijono, joka on dialogipuun sarjoitettu versio. Kumouksen tapahtuessa tämä sarjoitettu merkkijono muutetaan taas dialogipuuksi. Kumoustilojen määrä on keinotekoisesti rajattu 128:een, jotta pitkien sessioiden aikana ei syntyisi suuria kumoamistilalistoja.

Kuvassa 28 ovat nähtävissä kaikkien yksittäisten elementtien muokkaukseen tarkoitettujen paneelien luokkakaaviot. DialogueTreeNodeEdit perii jälleen UserControl-luokasta, mutta aiheuttaa olemassaolollaan ongelmia, sillä Visual Studio ei anna muokata näitä paneeleja visuaalisesti tämän ”väliluokan” luomisen jälkeen. Tämän vuoksi luokka asetettiin välikappaleeksi vasta projektin ollessa lähes valmis. Useat elementeistä vaativat tietoa kaikista NPC-repliikeistä, minkä vuoksi jo DialogueTreeNodeEdit pitää hallussaan listaa niistä. Viittaukset NPC-repliikkeihin valitaan yleensä pudotusvalikosta, mutta DialogueOptionEdit tarjoaa myös etsinnän valintamenetelmänä.



Kuva 28. MainControls-luokan oikean paneelin täyttävät luokat.

Kuvassa 29 on esimerkkinä hahmoelementin arvojen muokkaamiseen tarkoitettu CharacterEdit. Kuvan mukaisesti ehto-repliikki parien järjestystä voidaan muokata, sillä ehdon täyttyessä repliikki päätetään ja siksi myöhempiä ehtoja ei tarkasteta. Pelaajan vastausvaihtoehtojen kohdalla kaikkien ehtojen pitää täyttyä, joten niiden järjestyksellä ei ole väliä.



Kuva 29. Esimerkki MainControls-kontrollin oikeanpuoleisen paneelin elementtikohtaisesta muokkauskontrollista.

ConditionIDPairEdit on nimen mukaisesti kontrolli, jonka avulla muokataan ConditionIDPair-instanssien arvoja. Tätä käyttöliittymän palasta on yhtä monta kuin muokattavalla hahmolla on listassaan ehto-repliikkipareja, ja vaikkei erillistä luokkaa välttämättä olisikaan tarvinnut, se sievensi koodia huomattavasti. FieldEdit palvelee hyvin samankaltaista tarkoitusta, sillä se ei myöskään ole ehdoton, mutta koska sekä dialogivaihtoehdoilla että NPC-repliikeillä voi olla useita toimintoja ja dialogivaihtoehdoilla myös ehtoja, on sen olemassaolo koodin erottelu- ja sievennysmielessä hyödyllinen.

5 Testaus

5.1 Suorituskykytestaus

Hitaimmat operaatiot dialogieditorissa ovat projektitiedostojen avaus ja sellaisten muokkauspaneelien avaus, joissa valitaan NPC-repliikkejä pudotusvalikosta.

Projektitiedoston avauksen nopeutta testattiin testitiedostoilla, jotka sisälsivät suuren määrän solmuja. Testitiedostossa useita kertoja toistuva solmurypäs alkaa yksittäisellä

hahmosolmulla. Tämän solmun alla on yksi NPC-repliikki, jonka alla taas on kolme pelaajan dialogivaihtoehtoa. Tätä viiden solmun rypästä kopioitiin tiedoston sisällä saavuttaen lopulta tarvittava määrä. Tätä solmurypästä testattiin ohjelman sisällä suorittamalla avausoperaatio 100 kertaa ja ajastamalla jokainen kerta C#:n StopWatch-luokan avulla.

Ensin testattiin vaatimuksessa ETVS02 mainittua 2 000 solmun projektia, jonka vaatimuksen mukaan pitää pystyä latautumaan alle sekunnissa. Tällaisen tiedoston keskimääräinen latausaika oli 33 millisekuntia, mikä on huomattavasti alle rajan.

Toiseksi kokeiltiin 10 000 solmun projektia, jossa keskiarvo nousi jo 173 millisekuntiin, mikä tarkoittaa lähes lineaarista kasvua 2 000 solmun tiedostoon verrattuna. 20 000 solmun tiedostossa keskiarvo oli 424 millisekuntia, ja 50 000 solmun tiedostossa rikottiin jo sekunnin raja, kun keskiarvo oli 1 342 millisekuntia. Ohjelma kestäisi ladata suurempiakin solmumääriä suhteellisen siedettävässä ajassa, mutta peleissä, jotka sisältävät mittavan määrän dialogia, ei kaiken dialogin pakkaaminen yhteen tiedostoon ole järkevää. 50 000 solmun testitiedosto oli jo kooltaan yli 6,5 megatavua ja sisälsi yli 200 000 riviä.

Pudotusvalikon muodostus esimerkiksi DialogueOptionEdit-kontrollin rakentamisen yhteydessä aloittaa itse asiassa hieman hitaampana kuin itse projektin lataus, mutta hidastuu solmuja lisättäessä vähemmän. 2 000 solmun projektissa kesto on keskimäärin 46 millisekuntia, 10 000 solmun projektissa 122 millisekuntia ja 50 000 solmun kanssa 487 millisekuntia.

Datan lukeminen itsessään ja Root-luokan instanssin muodostus siitä ei kestä latausajasta kuin murto-osan, mutta graafisten elementtien muodostus on pullonkaulana. Nopeuden vuoksi TreeView on supistettuna eikä laajennettuna latauksen jälkeen.

5.2 Yksikkötestaus ja staattinen testaus

Huomattava osa niin sanotusta liiketoiminnallisesta kerroksesta dialogieditorissa on kiedottu graafisen käyttöliittymän palasiin, minkä vuoksi yksikkötestien luominen kattavaa lähestyvässä määrin olisi hankalaa, mahdotonta tai huomattavaa uudelleenjärjestelyä vaativaa. Dialogipuuelementtiluokat itsessään ovat suurimmaksi osaksi taas hyvin eriteltyjä ja niiden yksikkötestaus olisi helppoa, mutta nämä luokat ovat varsin yksinkertaisia, ja yksikkötestauksen hyödyt jäisivät kovin rajallisiksi.

Mikäli projektia olisi lähestytty testivetoisesti ja alusta alkaen olisi luotu yksikkötestejä, niistä olisi voinut saada merkityksellistä hyötyä, varsinkin jos olisi pyritty noudattamaan tiukasti esimerkiksi MVC-mallia (Model-View-Controller), mutta projektin pieni koko huomioon ottaen, tämä olisi mahdollisesti tuottanut enemmän vaivaa kuin hyötyä.

Staattista testausta projekti sallii nykyisessä muodossaankin, ja Visual Studio tarjoaa omat työkalunsa tähän, kuten kuva 30 osoittaa.

Code Metrics Results

Filter: None Min: Max:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
DialogueEditor (Debug)	76	410	8	112	1 509
{ } DialogueEditor.SubUserControls	68	25	7	24	110
{ } DialogueEditor.DialogueTreeNodes	89	56	2	10	81
Root	75	12	2	6	22
NPCLine	89	15	2	5	21
DialogueTreeNode	92	7	1	2	9
DialogueOption	91	14	2	3	17
Custom	95	3	2	1	3
Character	90	5	2	3	9
{ } DialogueEditor	72	329	8	111	1 318

Kuva 30. Koodimetriikkatuloksia Visual Studion sisällä.

Käyttöliittymäpalasten ylläpitoindeksiä (Maintainability Index) vetävät alas muun muassa Visual Studion automaattisesti luodut metodit, etenkin InitializeComponent(), jossa alustetaan kaikki kontrollin komponentit. Dialogipuuelementtiluokat ovat hyvin lähellä 100:aa, mikä on odotettavissa jo niiden yksinkertaisuus huomioon ottaen. Microsoftin mukaan numero välillä 20–100 tarkoittaa hyvää ylläpitävyyttä, joten se, että jokainen ylätason luku on vähintään 68, on hyvä asia (45).

Koodinmäärän suhteen MainWindow ja MainControls ovat selkeästi suurimpia 263 ja 258 rivillä koodiaan. Monissa käyttöliittymäelementeissä on syvä perintäketju, mutta sille ei voi paljon mitään, sillä esimerkiksi Windows Forms -luokka UserControl itsessään on jo muutaman perinnän takana.

5.3 Vaatimusvastaavuus

Ei-toiminnallisista vaatimuksista ainoastaan ETVS01 jää täyttämättä, sillä pudotusvalikoiden täyttö NPC-repliikeillä kestää joskus suurissa projekteissa kauemmin kuin 0,2 sekuntia.

Toiminnallisissa vaatimuksissa on useita puutteita:

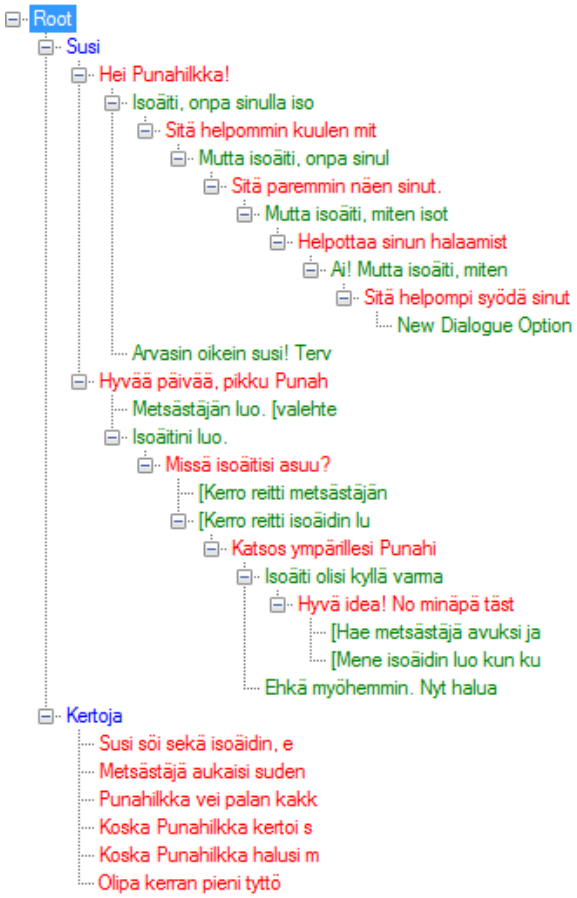
Prioriteetin 2 vaatimuksista, eli tarpeellisista vaatimuksista, ainoastaan TV24:ssä on puutteita ja TV27 puuttuu kokonaan. Vaikka ohjelmassa pystyy leikkaamaan ja liimaamaan solmuja, ei kopiointi onnistu. Tämä tehtiin siksi, että syytä kopiointiin ei pitäisi olla ja tunnisteiden täytyy olla ainutlaatuisia. TV27:n ajatus samantyyppisten solmujen sisäkkäisyyden estämisestä ei käytännössä ole hyvä ajatus, sillä on tilanteita, joissa sellaista sisäkkäisyyttä vaaditaan.

Toivottavissa, eli 3 prioriteetin, vaatimuksissa on paljon puutteita, mikä oli odotettavissa vaatimuksia päätettäessä.

TV12, TV18 ja TV19 ovat vaatimuksia, joitten voisi sanoa täyttyvän osittain. Tai pikemminkin vaatimukset voi toteuttaa repliikkien toiminnoissa, mutta yhtään niistä ei ole eksplisiittisesti saatavilla editorissa. Vaatimuksen 25 erillisestä näkymästä dialogin kulun selvittämiseksi todettiin myöhemmin prosessissa olevan turha, sillä TreeView-näkymä tarjoaa jo hyvän näkymän itsessään. Vaatimukset TV07, TV26, TV30 ja TV31 jäivät toteuttamatta. TV26-vaatimuksen mukainen käsikirjoitusnäkymä olisi ollut mahdollista toteuttaa rajattuna versiona, mutta sen toteuttaminen täysin varustettuna olisi vaatinut enemmän muuttujia ja mahdollisesti muun muassa paikan pakollisena ylätason solmuna. Vaatimus päätettiin näin ollen hyllyttää. Mahdollisuus testata dialogia ohjelman sisällä vaatimuksen TV31 mukaisesti olisi ollut mahdollinen toteuttaa, ja se on ominaisuus, joka kilpailevissa tuotteissa on, mutta sen hyödyt ovat rajalliset suhteessa sen vaatimaan vaivaan. TV07 toteutettiin osittain siinä mielessä, että ohjelma muistaa viimeksi avatun kansion. TV30:n solmuetsintä jätettiin toteuttamatta yksinkertaisesti sen rajallisen arvon vuoksi.

5.4 Testipeli

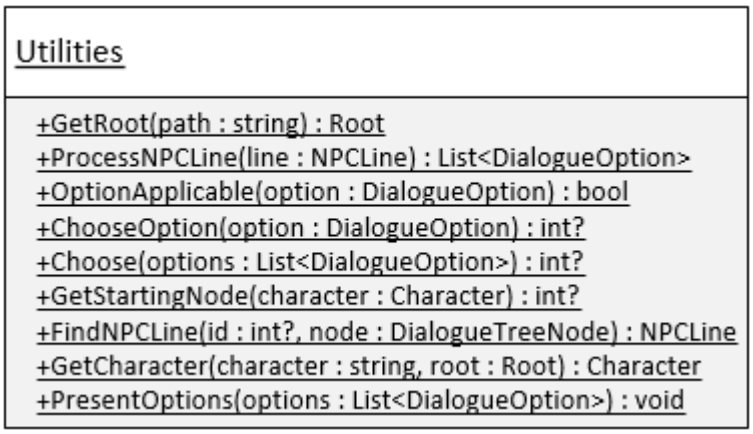
Dialogieditorin testaamiseksi käytännössä toteutettiin pieni peli, jota voisi luonnehtia tekstiseikkailuksi, joskin pelaaja ei tee dialogin ulkopuolella käytännössä mitään. Pelin dialogi luotiin dialogieditorin avulla, ja tarkoituksena oli käyttää jokaista erillistä editorin tarjoamaa datatiedostoon menevää ominaisuutta lukuun ottamatta räätälöityjä solmuja. Tekstiseikkailu on lyhyt, noin muutaman kymmenen solmun Punahilkka-sadun mukaelma (kuva 31).



Kuva 31. Tekstiseikkailu dialogieditorissa.

Valmiin dialogitiedoston lukemista olisi voinut lähestyä kahdella tavalla: joko poistamalla sarjoitus, mikä tarkoittaa DialogueTreeNode-luokkien läsnäoloa myös tekstiseikkailussa, tai sitten lukemalla tiedostoa kuin mitä tahansa muuta XML- tai JSON-tiedostoa ilman sarjoituksen poistoa. Tässä tapauksessa valittiin sarjoituksen poisto menetelmäksi.

Käytännössä datan kannalta oleellisia luokkia tässä tekstiseikkailussa ovat staattinen luokka Utilities, joka sisältää muun muassa repliikkien etsimiseen tarvittavan metodin (kuvan 32 UML-luokkakaavio), ja luokka State, joka on niin sanottu ”Singleton”-luokka ja sisältää yksinkertaisen listan avain-arvopareja C#:n Dictionary-luokkaa hyödyntäen. Näissä avain-arvopareissa avain on string-tyyppiä ja arvo taas totuusarvo eli bool-tyyppinen. Avaimet ovat samoja kuin mitä dialogieditorissa asetettiin erinäisissä ehto- ja toiminta-kohdissa. Näin esimerkiksi, kun jonkin repliikin kohdalla esiintyy toiminto ”end”, tiedetään tässä esimerkissä, että tarkoituksena on muuttaa avaimen ”end” arvo totuudeksi.



Kuva 32. Staattisen Utilities-luokan UML-luokkakaavio.

Voi olla, että todennäköisempi käyttökohde toiminnoille ja ehdoille repliikkien yhteydessä olisi esimerkiksi jonkinlaiset skriptit, mutta tässä esimerkkiohjelmassa niitä käytetään totuusarvojen testaamiseen ja muuttamiseen, sillä skriptaus itsessään ei ole olennaista tässä insinöörityössä.

Seuraavaan repliikkiin siirtyminen sujui pitkälti suunnitelmien mukaan: kun on kyse NPC:n repliikistä, sen alta löytyvät välittömästi pelaajan vastausvaihtoehdot, ja vastaavasti, kun on kyse pelaajan repliikistä, seuraavaan repliikkiin on viite muuttujassa. Tämä on kuitenkin jokseenkin jäykkä menetelmä, mikä oli tiedossa jo etukäteen. Suurin osa dialogista noudattaa tätä edestakaisin-rakennetta, mutta kun poikkeamia tapahtuu, joutuu dialogieditorissa tekemään asioita, jotka eivät välttämättä ole itsestään selviä.

Tässä pelissä yksi tämäntyyppinen poikkeama esiintyy, kun NPC-repliikistä halutaan siirtyä toiseen NPC-repliikkiin eikä kyseistä NPC-repliikkiä ollut loogista sijoittaa suoraan ensimmäisen alle, sillä se kuului eri hahmolle. Tällaisissa tapauksissa on ainakin kaksi ratkaisua: joko repliikkiviitteen sijoittaminen toiminnaksi ensimmäiseen NPC-repliikkiin

tai tyhjän pelaajadialogivaihtoehdon käyttäminen pelkkänä linkkinä seuraavan NPC-repliikkiin kahden NPC-repliikin välissä. Dialogieditoriin olisi voitu sijoittaa valinnainen NPC-repliikkiviitemuuttuja myös NPC-repliikkeihin, mutta tämä olisi omalta osaltaan tehnyt editorin käytöstä hieman sekavampaa, minkä vuoksi tällaista vaihtoehtoa ei ohjelmassa ole.

Koska tunnisteet repliikeissä ovat uniikkeja, ei tästä seuraa suurta vaivaa, mutta jos viittauksen yhteydessä olisi tieto myös hahmosta, olisi repliikin löytäminen nopeampaa. Tämä johtaisi kuitenkin jälleen ylimääräiseen tiedonsyöttöön editorissa, ja näin ollen jokainen viittaus veisi hieman enemmän aikaa tekoprosessissa.

Syy, miksi ohjelmassa ylipäätään on käytössä järjestelmä, jossa NPC-repliikit ja pelaajan repliikit eivät ole samantyyppisiä, on se, että tällä tavoin poistetaan tarve valita repliikin puhuja joka kerta, kun uusi repliikki luodaan. Ylimääräisten klikkausten ja valintojen poisto on toisin sanoen tavoitteena.

Räätälöityjä solmuja ei pelissä ole käytössä, mutta niitten käyttö aiheuttaisi hieman päänvaivaa, sillä niitä voi sijoittaa minne tahansa ja niiden alle voi sijoittaa mitä tahansa. Tämä tarkoittaa käytännössä sitä, että dialogieditorin käyttäjä voi luoda räätälöityjen solmujen avulla täysin epäloogisen rakenteen, mutta tämä on ongelma, joka seuraa suoraan kyseisen solmun vapaudesta, ja sen vapauden rajoittaminen taas tarkoittaisi, että solmun käyttökohteet kapenisivat.

Testipelin luonti todisti, että perusominaisuudet toimivat ilman ongelmia, mutta testipeli paljasti myös dialogieditorin puutteita. Editorin ominaisuudet rohkaisevat käyttäjää käyttämään tietynlaista muottia dialogin rakenteessa, ja tästä muotista poikkeaminen tarkoittaa usein epäintuitiivisten ratkaisujen käyttämistä. Näitä poikkeuksia ei kuitenkaan testipelin aikana tapahtunut paljon ja tavallisessa dialogissa ne tulevat olemaan harvinaisia.

6 Yhteenveto

Ohjelma valmistui aikataulussa, ja valmis tuote täyttää lähes kaikki sille asetetut vaatimukset. Vakavia puutteita ei ohjelmassa ole, mutta siitä tuli odotettua jäykempi mahdol-

listen dialogirakenteiden suhteen. Tämä jäykkyys paljasti syyn monille kilpailevien tuotteiden ensisilmäykseltä tehottomille ratkaisuille, ja on mahdollista, että mikäli tuotetta alettaisiin kehitellä uudestaan, tehtäisiin dataelementtien osalta asioita toisin.

Jatkokehityksen kannalta sovellus kaipaisi mahdollisesti ohjekirjaa valaisemaan ominaisuuksia ja mahdollisuuksia, jotka eivät ole itsestään selviä, mutta yleisimmin käytettävät ominaisuudet ovat ohjelman sisällä suurimmaksi osaksi selitettyjä.

Toinen asia, joka olisi ollut toivottava, on kirjasto, joka toimisi välikätenä pelin ja dialogieditorin tuottamien datatiedostojen välillä. Testipelin ohessa luotiin tietynasteinen prototyyppi, mutta itsenäistä kirjastoa ei ole, ja testipelin versio toimisi vain C#-kielisissä peleissä.

Sovellusta olisi voitu myös alkaa levittää verkossa esimerkiksi GitHubin kaltaisessa palvelussa, jonne projektin voi laittaa julkisesti näkyväksi. Insinöörityön osalta tätä kuitenkin pidettiin epäolennaisena.

Ohjelman valmistus sujui nopeasti, ja jo olemassa ollut ymmärrys valituista työkaluista auttoi merkittävästi kehitystä. Windows Forms -kirjasto oli ennestään tuttu, mutta projekti opetti kirjastosta paljon lisää.

Yleisesti ottaen dialogieditori ja sen tuottamat datatiedostot toimivat suunnitellulla tavalla, ja lopputulosta voi kuvailla onnistuneeksi. Ongelmia on, mutta editorin ominaisuudet ovat riittävän joustavia mahdollistamaan yleensä ongelmien kiertämisen, ja perusominaisuudet toimivat moitteetta.

Lähteet

1 Galyonkin, Sergey. 2016. Steam Sales in 2015. Verkkodokumentti. Medium. <https://medium.com/steam-spy/steam-sales-in-2015-2e81a6bb0f5a>. 4.1.2016. Luettu 28.1.2016.

2 Wilson, Robin J. 1996. Introduction to Graph Theory. 4. painos. Harlow, Essex: Addison Wesley Longman.

3 Weller, Vince D. 2015. Age of Decadence. Iron Towers Studio.

4 Finite State Machine. Verkkodokumentti. MathWorks. <http://se.mathworks.com/discovery/finite-state-machine.html?s_tid=gn_loc_drop>. Luettu 29.1.2016.

5 Sipser, Michael. 1997. Introduction to the Theory of Computation. Boston: PWS Publishing Company.

6 Dialogue | Definition of Dialogue by Merriam-Webster. Verkkodokumentti. Merriam-Webster. <http://www.merriam-webster.com/dictionary/dialogue>. Luettu 25.1.2016.

7 Weizenbaum, Joseph. 1966. ELIZA – A Computer Program for the Study of Natural Language Communication Between Man and Machine. Communications of the ACM. Volume 9, number 1.

8 Colossal Cave Adventure. Verkkodokumentti. GiantBomb. <http://www.giantbomb.com/colossal-cave-adventure/3030-19301/>. Luettu 25.1.2016.

9 Garriott, Richard. 1985. Ultima IV: Quest of the Avatar. Origin Systems.

10 Alice in Wonderland. 1985. Windham Classics.

11 Starflight. 1986. Electronic Arts.

12 The Secret of Monkey Island. 1990. LucasArts.

13 Graphical Interface for Interactive Dialog. Verkkodokumentti. United State Patent and Trademark Office. <http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=8,082,499.PN.&OS=PN/8,082,499&RS=PN/8,082,499>. Luettu 3.2.2016.

14 Mass Effect. 2007. Electronic Arts.

15 Indigo Prophecy / Fahrenheit. 2005. Atari.

16 Seawalker, Ben. 2015. Chronicler – ChoiceScript Visual Code Editor. Verkkodokumentti. <https://forum.choiceofgames.com/t/tool-chronicler-choicescript-visual-code-editor/6811>. Luettu 20.1.2016.

17 Coculuzzi, Tony. Dialoguer. Verkkodokumentti. <http://www.dialoguer.info/>. Luettu 20.1.2016.

18 Dialogue System for Unity. Verkkodokumentti. Pixel Crushers. <http://www.pixelcrushers.com/dialogue-system/>. Luettu 20.1.2016.

19 Rustybulletholes. Basic Dialogue Editor. Verkkodokumentti. Unity. <https://www.assetstore.unity3d.com/en/#!/content/36885>. Luettu 20.1.2016.

20 Articy:draft overview. Verkkodokumentti. Nevigo. <http://www.nevigo.com/en/articydraft/overview/>. Luettu 19.1.2016.

21 Chat Mapper. Verkkodokumentti. Chat Mapper. <http://www.chatmapper.com/>. Luettu 19.1.2016.

22 TalkerMaker Deluxe. Verkkodokumentti. Digiwombat. <https://github.com/digiwombat/TalkerMakerDeluxe>. Luettu 19.1.2016.

23 Fitzgerald, Randall. 2015. Right Tools: TalkerMaker Deluxe. Verkkodokumentti. <http://www.barkyseal.com/2015/04/27/the-right-tools-talkermaker-deluxe/>. 27.4.2015. Luettu 19.1.2016.

24 Choose Your Technology. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/library/windows/desktop/dn614993.aspx>. Luettu 20.1.2016.

25 Sonmez, John. 2010. Simple Programmer. Verkkodokumentti. <http://simpleprogrammer.com/2010/02/08/c-vs-java-part-2-the-platforms-desktop-and-mobile/>. 8.2.2010. Luettu 20.1.2016.

26 YAML Ain't Markup Language (YAML™) Version 1.2. Verkkodokumentti. The Official YAML Web Site. <http://yaml.org/spec/1.2/spec.html#id2759572>. 1.10.2009. Luettu 16.1.2016.

27 Google Trends. Verkkodokumentti. Google. <https://www.google.fi/trends/explore#q=XML%2C%20JSON%2C%20YAML&date=1%2F2006%20121m&cmpt=q&tz=Etc%2FGMT-2>. Luettu 13.1.2016.

28 PYPL PopularitY of Programming Language. Verkkodokumentti. PYPL. <http://pypl.github.io/PYPL.html>. Luettu 13.1.2016.

29 Stack Overflow. Verkkodokumentti. Stack Overflow. <http://stackoverflow.com/tags>. Luettu 13.1.2016.

30 API Directory. Verkkodokumentti. ProgrammableWeb. <http://web.archive.org/web/20131226092041/http://www.programmableweb.com/apis/directory>. Luettu 14.1.2016.

31 JSON's Eight Year Convergence With XML. Verkkodokumentti. ProgrammableWeb. <http://www.programmableweb.com/news/jsons-eight-year-convergence-xml/2013/12/26>. Luettu 14.1.2016.

32 XmlSerializer Class. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/system.xml.serialization.xmlserializer%28v=vs.110%29.aspx>. Luettu 16.1.2016.

33 Json.NET vs .NET Serializers. Verkkodokumentti. Newtonsoft. <http://www.newtonsoft.com/json/help/html/JsonNetVsDotNetSerializers.htm>. Luettu 16.1.2016.

34 Extensible Markup Language (XML) 1.0 (Fifth Edition). Verkkodokumentti. W3C. <https://www.w3.org/TR/xml/>. Luettu 11.1.2016.

35 Guinness World Records 2011 – Gamer's Edition. 2011. DK Publishing.

36 Retrospective: Planescape Torment. 2009. Verkkodokumentti. Eurogamer. <http://www.eurogamer.net/articles/planescape-torment-retrospective>. 23.8.2009. Luettu 11.1.2016.

37 Average Word Length in the English Language. Verkkodokumentti. Wolfram Alpha. <http://www.wolframalpha.com/input/?i=average+english+word+length>. Luettu 12.1.2016.

38 The Problem with Growing Download Sizes. 2015. Verkkodokumentti. PCGamer.<http://www.pcgamer.com/the-problem-with-growing-download-sizes/>. 14.4.2015. Luettu 12.1.2016.

39 Bod, Damien. 2014. Comparing Protobuf, JSON, BSON, XML with .NET for File Streams. Verkkodokumentti <http://damienbod.com/2014/01/09/comparing-protobuf-json-bson-xml-with-net-for-file-streams/>. 9.1.2014. Luettu 15.1.2016.

40 Ganshani, Punit. 2013. Which One is Better: JSON vxML Serialization. Verkkodokumentti. <http://www.codetails.com/2013/10/23/which-one-is-better-json-vs-xml-serialization/>. 23.10.2013. Luettu 15.1.2016.

41 XML Essentials. Verkkodokumentti. W3C. <https://www.w3.org/standards/xml/core>. Luettu 17.1.2016.

42 Json.org. Verkkodokumentti. <http://www.json.org/>. Json.org. Luettu 10.1.2016.

43 IEEE Standard Glossary of Software Engineering Terminology. 1990. Verkkodokumentti. IEEE. <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=159342&filter=AND%28p_Publication_Number:2238%29>. Luettu 16.4.2016.

44 Serialization (C# and Visual Basic). Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/ms233843.aspx>. Luettu 20.2.2016.

45 Code Metrics Values. Verkkodokumentti. Microsoft. <https://msdn.microsoft.com/en-us/library/bb385914.aspx>. Luettu 29.3.2016.